Telegram Bot: Track and Field

Jesús De Oro García
Héctor Gálvez Bernal
Emilio Chico Muñoz

Director:
Carlos Gregorio Rodríguez

GRADO EN INGENIERÍA INFORMÁTICA

FACULTAD DE INFORMÁTICA

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO FIN DE GRADO
Junio 2017

Resumen

Todo el mundo está conectado, ya sea a través de su teléfono móvil, *tablet*, *Smart TV* o cualquier tipo de ordenador, por eso, decidimos buscar un Trabajo Final de Grado que nos permitiese explorar estas condiciones en las que hoy en día vivimos. Nos decantamos por un trabajo que tuviera relación con las aplicaciones de mensajería y los juegos, que conecte a la gente, entretenga y tenga posibilidades de futuro.

Resulta curioso ver la cantidad de juegos que hay en las tiendas de aplicaciones, pero son muy pocos los que existen integrados en las aplicaciones de mensajería. Esto es debido a que la mayoría están centrados en las redes sociales y en las grandes plataformas de videojuegos.

Nuestro proyecto consiste en la creación de un juego dentro de la aplicación de mensajería multiplataforma Telegram, concretamente hablamos del desarrollo de un *bot* que nos permita jugar en cualquier momento con nuestros amigos. El tema elegido para el juego, ha sido la prueba de atletismo de los 100 metros lisos.

Para lograr nuestro fin necesitamos una aplicación que transfiera datos entre el cliente y el servidor de manera instantánea, para dar sensación de dinamismo y realismo al juego. Con este objetivo en mente, hemos desarrollado dos *web services*, el primero para hospedar el *bot* en Telegram, el cual reúne la información necesaria del usuario y el segundo para gestionar las conexiones del juego, conexiones que son únicas en función a los datos personales del usuario obtenidos del primer *web service*.

El único requisito que se necesita para jugar es tener instalado Telegram actualizado a su última versión. Una vez dentro de la aplicación, basta con abrir una conversación entre cualquiera de tus amigos o grupos, e invocar al *bot* mediante el comando requerido.

En cuanto a modos de juego, disponemos de modo *Single,* para jugar contra la propia inteligencia artificial, el cual proporciona una serie de bots rivales que competirán contra el jugador.

En el modo *Multiplayer* podremos jugar contra cualquier persona del mundo que esté jugando en ese momento al juego, y finalmente, considerando que es la opción más social, está la opción *Group*, donde podremos jugar contra los miembros de tu grupo de amigos. Podemos empezar la partida cuando los usuarios dispuestos a jugar estén listos, sin tener que esperar a los demás indecisos.

Para terminar, también disponemos de un tutorial para hacer al usuario con las funcionalidades de la aplicación.

Como conclusión podemos decir, y estar orgullosos de haber aceptado el reto de crear uno de los primeros juegos multijugador en tiempo real en una plataforma de mensajería, con el objetivo de pasar un rato agradable con nuestros amigos desde la distancia.

Abstract

Everyone is connected, whether through their mobile phone, tablet, Smart TV or any kind of computer, so we decided to look for a Bachelor Final Project that could allow us to explore these world in which we live in. We opted for a project related to messaging applications and games, to connect people, entertain and have possibilities for the future.

It is interesting to see the amount of games that are in the application stores, but very few of them are integrated into messaging applications. This is because most of them are focused on social networks or on big video game platforms.

Our project consists on the creation of a game within the multiplatform messaging application Telegram, we are talking about the development of a bot that allows us to play at any time with our friends. The chosen theme for the game is the 100 metres race from the track and field competitions.

To achieve our goal, we need an application that transfers data between the client and the server instantly, to give this sense of dynamism and realism to the game. Keeping this goal in mind, we have developed two web services, the first one to host the bot in Telegram, which gathers the necessary information of the user and the second one to manage the connections of the game, connections that are unique depending on the personal information provided from the first web service.

The only requirement you must meet to be able to play the game is having Telegram updated to the latest version. Once inside the application, simply open a conversation between any of your friends or groups, and invoke the bot using the required command.

In terms of game modes, we have a Single mode, to play against the artificial intelligence, which provides a series of rival bots that will compete against the player.

In the Multiplayer mode, we can play against any person in the world who is playing the game at that moment, and finally, considered for us to be the most social mode, is the Group option, where we can play against the members of our group of friends. The game can be started when the rest of users willing to play are ready, without having to wait for the undecided ones.

Finally, we also have a tutorial to get the user through the features of the application.

As a conclusion, we are proud we accepted the challenge of creating one of the first real-time multiplayer games on a messaging platform, with the goal of having a quality and comfortable time with our friends in the distance.

Palabras clave

- Bot
- Telegram
- Websockets
- HTML5
- Motor Gráfico
- Javascript
- Node.js

Keywords

- Bot
- Telegram
- Websockets
- HTML5
- Graphic Engine
- Javascript
- Node.js

Índice

Índice de Ilustraciones

Capítulo 1: Introducción

El propósito de este primer capítulo, es dar a conocer la motivación que nos ha llevado a desarrollar este proyecto, y explicar su estrecha relación con la sociedad en la que vivimos y las tecnologías actuales.

El uso de dispositivos móviles está creciendo a un ritmo elevado [1], y con ello, el uso de las redes sociales y la mensajería instantánea [2]. Pues bien, este tipo de mensajería no siempre se utiliza como medio de comunicación, sino que también posee un gran abanico de posibilidades entre ellas el mundo de los juegos integrados.

Motivaciones

Actualmente existen millones de aplicaciones con diferentes finalidades, ya sea para realizar una tarea básica como apuntar una nota hasta poder controlar satélites a millones de kilómetros de la tierra.

Sin duda hay aplicaciones para todos los gustos, pero nos centramos en las aplicaciones de mensajería, ya que vivimos en un mundo saturado por la información y donde estar conectado y disponible es esencial [3]. Las aplicaciones de mensajería sirven para mantener el contacto mediante mensajes de texto principalmente, pero es cierto que también existen llamadas y video llamadas. También hay una gran variedad de juegos ya sea por descarga directa o mediante aplicaciones como *Google Play*. ¿Y si pudiésemos mezclar ambos mundos, la mensajería instantánea y los juegos?

Hoy en día estos servicios de mensajería están en continua expansión, siendo de las aplicaciones más descargadas, pero hasta hace muy poco, no existía el desarrollo por parte de terceros de juegos para estas aplicaciones, por lo que optamos por elaborar un juego con el que poder competir con varios integrantes dentro de una conversación en tiempo real.

Una de estas aplicaciones de mensajería es Telegram, en la cual hace poco se apostó por el desarrollo de terceros y así poder acceder al desarrollo de juegos.

El hecho de poder ser de los primeros en elaborar un juego multijugador en tiempo real dentro de esta aplicación supuso una gran motivación, además de tratarse de un reto, ya que hay muy poca información disponible acerca de este tema.

Nos referimos a los llamados *bots* con los que podemos acceder a juegos añadiendo un simple comando dentro de una conversación, lo cual explicaremos más adelante en este documento.

Introduction

The purpose of this first chapter is to show the motivations that led us to develop this project and explain its close relationship with the society in which we live in with the current technologies.

The mobile devices usage is increasing at a high rate, and with it, the social networks and instant messaging usage as well. This type of messaging is not always used as a way of communication, but also has a wide range of possibilities including the world of integrated games. REVISAR

Motivations

Currently there are millions of applications with different purposes, either to perform a basic task like saving a note to control satellites millions of miles from the earth.

There are certainly applications for all tastes, but we are focusing on messaging applications, as we live in a world overwhelmed by information in which being connected and available is essential. Messaging applications are used to maintain contact through text messages mainly, but it is true that there are also calls and video calls. There is also a vast variety of games either by direct download or by applications stores such as Google Play. What if we could mix both worlds, instant messaging and games?

Nowadays these messaging services are in continuous expansion and are some of the most downloaded applications, but until recently there was no third-party development of games for these applications, because of that we chose to create a game in which you could compete against several users within a real-time conversation.

Telegram is one of these messaging applications, in which recently, support for third party application development was added, and therefore allowing game development too.

The fact of us being one of the first teams developing a multiplayer game in real time within this application was a great motivation, as well as a challenge, since there is really scarce information about this matter.

We are referring to the bots, with them we can access a variety of games by adding a simple command inside a conversation, all of this will be explained later on this document.

Capítulo 2: Planteamiento del trabajo

En este segundo capítulo vamos a hablar acercas de los objetivos prioritarios sobre los cuales hemos realizado nuestro trabajo, así como la organización y estructura del mismo.

Objetivos

Nuestros principales objetivos en este proyecto son:

- Búsqueda y comparativa de juegos incorporados en aplicaciones de mensajería.
- Aprender el uso de *sockets*, *web services* y motores gráficos.
- Desarrollar uno de los primeros juegos totalmente online para Telegram.
- Crear un código modular, para que pueda ser fácilmente ampliable.

Estructura del trabajo

Comenzamos la memoria con una breve introducción y las motivaciones que nos han llevado a hacer este proyecto durante el curso, ***primer capítulo***. Los objetivos que nos hemos marcado para su completo desarrollo, y como hemos estructurado nuestro trabajo, es lo que constituye nuestro ***segundo capítulo***.

Para empezar el proyecto, es necesario conocer las tecnologías existentes y el contexto sobre el que lo vamos a desarrollar, por lo que en el ***tercer capítulo***, vamos a realizar un estudio sobre las diferentes tecnologías que podíamos aplicar a nuestro proyecto, empezando por las distintas aplicaciones de mensajería, seguido de los lenguajes de programación y motores gráficos y concluyendo con las herramientas disponibles de desarrollo, explicando en cada apartado cuales han sido nuestras elecciones para este proyecto.

En el ***cuarto capítulo*** vamos a describir con detalle el ámbito del sistema, las funcionalidades de la aplicación y haremos una exploración de la misma a través de un caso de uso completo, para finalizar se enumeran los requisitos necesarios por el usuario y el servidor para poder ejecutar la aplicación.

Durante el ***quinto capítulo*** nos centramos en la arquitectura del juego, separando claramente las distintas partes que lo forman, y explicando con rigurosidad el funcionamiento de cada una de una de ellas mediante un diagrama donde podemos ver todos los estados que forman la aplicación.

A lo largo del proyecto nos encontramos con problemas y dificultades que nos llevaron más tiempo del esperado solucionar, estos problemas y soluciones están reflejados en el ***sexto capítulo***.

A continuación, en el ***séptimo capítulo,*** se presentan las conclusiones a las que hemos llegado tras finalizar el proyecto, un proyecto con visión de futuro y posibilidades de expansión y mejoras, recogidas en el ***octavo capítulo***, donde se describen algunas de las posibles ampliaciones aplicables al proyecto en un futuro.

En el ***noveno capítulo*** exponemos las contribuciones personales de cada miembro del grupo en un formato de entrevista, así como el punto de vista y las sensaciones que hemos ido percibiendo según avanzaba el proyecto.

Para finalizar esta memoria se añaden dos ***anexos***, el primero incluye las fuentes de las referencias indicadas en el resto de capítulos de la memoria, y el segundo la bibliografía de la que nos hemos nutrido para explicar las distintas tecnologías expuestas en el tercer capítulo y las documentaciones que nos han permitido desarrollar el código de la aplicación.

Capítulo 3: Estudio de Tecnologías

En este capítulo vamos a exponer los diferentes estudios previos que hemos realizado sobre los métodos y tecnologías disponibles, analizándolos y concluyendo cual hemos decidido utilizar.

Aplicaciones de mensajería.

Actualmente las aplicaciones de mensajería son las más descargadas y las que más gustan a los usuarios. Pero antes de empezar a desarrollar, necesitamos saber cuál es la aplicación óptima para nuestro proyecto, necesitamos investigar bien sobre cómo funcionan los diferentes tipos de aplicaciones de mensajería, y si tienen soporte para desarrollar *bots* y juegos ya que vamos a basar nuestro proyecto en ello.

Telegram

Telegram es una aplicación de mensajería centrada en la rapidez y seguridad de sus conversaciones. Es totalmente gratuita, y está disponible para una gran variedad de dispositivos. Funciona con varios sistemas operativos como Android, MacOS y Linux. Se lanzó entre los meses de agosto y octubre del año 2013 con millones de descargas y desde entonces los usuarios que utilizan esta aplicación siguen creciendo.

No podemos hablar de Telegram sin compararlo con WhatsApp, pues es la aplicación de mensajería reina, y entre ellas existen grandes diferencias. La principal ventaja de Telegram es que está basada en la nube, es decir, podemos acceder a todas las conversaciones desde varios dispositivos a la vez. También posee la capacidad de compartir un número ilimitado de fotos y videos, y nos da la posibilidad de almacenarlos en la nube en el caso de no querer guardarlos en la memoria del dispositivo. Otra característica de Telegram es que es de código abierto en el lado del cliente, esta plataforma invita a todos los desarrolladores a crear sus propias aplicaciones basadas en Telegram; sin embargo, con WhatsApp esto no es posible.

Uno de los principales atractivos de Telegram es la posibilidad de desarrollar *bots*. Llamamos *bot* a aquella aplicación que se ejecuta dentro de Telegram pudiendo interactuar con ellos enviando mensajes, y por medio de comandos desde las conversaciones. Existen multitud de *bots* usados para diferentes fines, los hay para recibir notificaciones, integrarse con otros servicios como GIF, imágenes, música, YouTube, juegos, etc. En resumen, un *bot* puede utilizarse para casi cualquier cosa.

WhatsApp

Con alrededor de 10.000 millones de usuarios, WhatsApp es la aplicación de mensajería más descargada en el mundo. Se trata de la aplicación chat por excelencia, que comenzó propagándose debido a ser un método de intercambio de mensajes más económico que los SMS. Cuenta con una interfaz sencilla que hace que sea muy rápida de utilizar por el usuario, ya que basta con solo pulsar sobre un contacto para establecer una nueva conversación. Nos garantiza una gran accesibilidad pues no se necesitan grandes conocimientos para poder usarla, lo que hace que esto sea una de sus características. Cada usuario se identifica mediante su número de teléfono móvil y en cuanto a la seguridad, es una de las prioridades utilizando un sistema cifrado para proteger las conversaciones de extremo a extremo.

Aunque existen aplicaciones desarrolladas para Whatsapp, no son legales e incumplen los términos y condiciones de la aplicación, por lo que no se puede añadir nada a las funcionalidades básicas de esta red de mensajería.

Line

Para muchos, Line es la gran alternativa a las demás aplicaciones de mensajería. Su principal característica recae sobre el uso de *Stickers* o pegatinas, es decir, emoticonos mucho más grandes con un aspecto visual mucho más definido y desarrollado, rozando lo dramático y haciéndolos más animados.

Line fue de las primeras en incorporar llamadas gratuitas y con bastante calidad, por lo que despuntó entre sus competidoras. Otra de las características que ofrece Line, es la posibilidad de hablar sin revelar tu número de teléfono, pudiendo hacerlo mediante códigos QR y otros métodos.

Los *bots* también han llegado a Line, y es por eso que ha decidido abrir su A*pplication Programming Interface* (a partir de ahora API) a desarrolladores ofreciendo 10.000 accesos gratuitos, con algunas restricciones debido a que se encuentran en período de pruebas. Hasta ahora solo existían *bots* creados por Line en la propia aplicación, pero con el tiempo se han ido desarrollando *bots* con diferentes funcionalidades como pagar, hacer una reserva en un restaurante u hotel, o incluso pedir un taxi.

Facebook Messenger

Facebook Messenger es una aplicación independiente de Facebook, compatible con IOS, Android y Windows. Esta aplicación permite intercambiar mensajes de texto con tus amigos de Facebook, pudiendo estar conectado al mismo tiempo desde PC y dispositivo móvil. Al igual que las demás aplicaciones también incluye llamadas gratuitas a través de la tecnología VoIP, voz a través de internet.

Los *bots* también se han integrado en Facebook. Existen un montón diferentes, tales como *testing bots*, *followings bots*, *traffic bots* y muchos más. Todos ellos se crean gracias a la plataforma *Botsify*, que además te permite sincronizarlo con la aplicación de Facebook.

Conclusiones

Hemos elegido utilizar la plataforma de Telegram para el desarrollo de nuestro *bot,* debido a que se trata de una aplicación de código abierto en cliente y queríamos poder tener la mayor información acerca de su funcionamiento para poder optimizar nuestro proyecto. Además, Telegram fue de los primeros en agregar el soporte para *bots* con lo que la tecnología y la documentación está más avanzada que en la competencia.

Lenguajes de programación

Para poder llevar a cabo nuestro proyecto tenemos que entender cómo funcionan los lenguajes de programación sobre los que se desarrollan Juegos para Telegram. En este apartado vamos a estudiar y analizar las posibilidades que nos ofrece cada uno para elaborar un *bot* completamente funcional.

HTML5

HyperText Markup Language (de ahora en adelante, HTML) se remonta al año 1980. Aquí ya existía un tipo de documentos denominados Hipertexto, pero eran demasiado primitivos y por eso no eran muy utilizados, salvo para algún enlace web. Entonces su inventor, el físico Tim-Berners-Lee, propuso un sistema de Hipertexto para compartir documentos. Una vez desarrollado su sistema, y junto con la ayuda del ingeniero de sistemas Robert Cailliau, fundaron la *World Wide Web*.

Desde su creación hubo varias propuestas para que HTML se convirtiera en un estándar. La primera fue en 1993 por el *Internet Engineering Task Force* (IETF), pero no fue hasta 1995 cuando se consiguió, bajo el nombre de HTML 2.0. Durante los siguientes años surgieron nuevas actualizaciones, como la 3.1 que incluía java y texto alrededor de las imágenes, pero fue con la 4.0 cuando lograron dar el gran salto, pues esta llegó a integrar hojas de estilo CSS, tablas, formularios y los conocidos *scripts*.

Llegamos a la actualidad. HTML5 es un elemento principal para el funcionamiento de *webs*. Se trata de un lenguaje encargado de ordenar y presentar el contenido de las páginas web, es decir, es el encargado de modificar el *layout* y su aspecto.

HTML funciona a través de etiquetas. Éstas son la parte encargada de que todos los navegadores interpreten el código. Esta última actualización hace que la interpretación sea más rápida y fácil debido a la incorporación de nuevas etiquetas, ahorrándonos el uso de otros productos como *flash*. Aquí podemos ver un ejemplo:

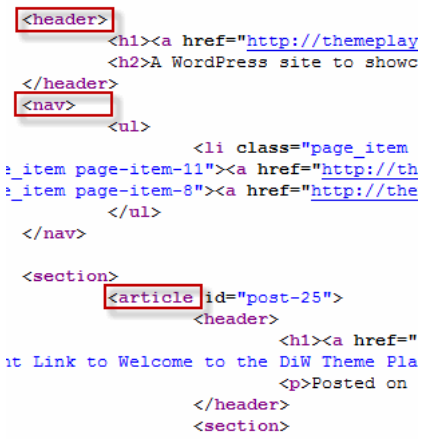

Imagen 1 - *Estructura HTML5*

Debido a que HTML5 solo actúa sobre el texto y el estilo, rápidamente surgió la necesidad de añadir animaciones. Según avanzamos en nuestro proyecto, vemos necesario el uso de *JavaScript* para proporcionar dinamismo al código HTML. Pues bien, *JavaScript* es un lenguaje de programación ejecutado en el lado del cliente.

El código siempre debe tener este formato:

```
<html>
	<head>
		<title>Embeber JavaScript – aprenderaprogramar.com</title>
	</head>
	<body>
		<script type="text/javascript">
			document.write('Hola Mundo');
		</script>
	</body>
</html>
```

Imagen 2 - *Ejemplo JavaScript*

Como vemos está entre las marcas:

`<script type="text/javascript"> </script>`

Este tipo de lenguaje es interpretado por el navegador y funciona mientras la página carga, pero solo sobre las acciones programadas en el código HTML.

A su vez todos los archivos de JavaScript son archivos de texto guardados con la extensión .js.

Node.js

Ryan Dahl, conocido ingeniero de Software crea en 2009 *Node.js* con la ayuda del motor V8 de Google, intentado compensar los malos hábitos de los desarrolladores a la hora de crear una nueva aplicación y que se vuelva inconsistente e inútil con el tiempo. Por ello, crea un entorno *JavaScript Node.js* que consiste en *JavaScript*, pero del lado del servidor, siguiendo la dinámica de la programación orientada a objetos.

Node.js funciona en una gran variedad de sistemas como Linux, Windows y MacOS. Una de las características más importantes de *Node.js* es que se trata de una librería asíncrona. La programación asíncrona consiste en devolver el control al programa llamante antes de que termine las funciones y así seguir operando en segundo plano; esto agiliza la ejecución y aumenta la escalabilidad.

Las funcionalidades de *Node.js* que más nos han llamado la atención son:

- **Gestión de paquetes**: *Node.js* cuenta con un gestor de paquetes preinstalado llamado *Node Package Manager* (NPM), que se encarga de instalar librerías y gestionar su dependencia desde la línea de comandos.

  Ponemos atención especial en este gestor de paquetes, pues nos resulta muy útil para conseguir que nuestro proyecto sea más liviano. NPM facilita la reutilización y actualización del código, lo que hace más ameno el trabajo para los desarrolladores. Utiliza el archivo *package.json* para la definición de librerías a instalar.

- **Bucle de eventos:** *Node.js* funciona a través del método bucle de eventos. Cuando ejecutas una aplicación se crea un hilo y queda a la espera de que alguien llegue y le haga una petición; una vez obtenida, ninguna otra puede ser procesada hasta que acabe el proceso actual.

Recordemos que *Node.js* trabaja de forma asíncrona, por lo que esto no le supone ninguna complicación; aquí entra el bucle de eventos. Para cada petición, *Node.js* añade una función *callback* a un evento de respuesta, entonces cuando el recurso queda libre para las peticiones, llama a dicha función y procesa cada petición.

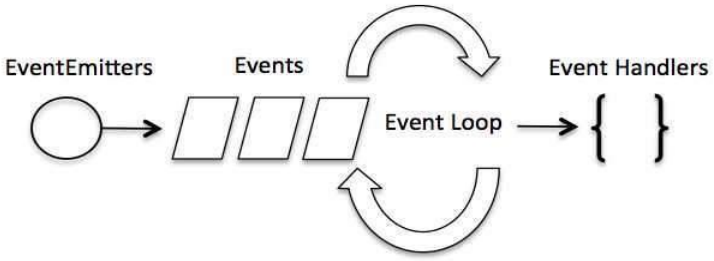

Imagen 3 - *Flujo de evento Node.js*

Motores gráficos

Un motor gráfico es un conjunto de librerías usadas por aplicaciones y programas para *renderizar* gráficos y sonidos de una forma más sencilla. Nuestro proyecto al basarse en un juego, necesita de un motor gráfico que se amolde a nuestra visión del juego.

Phaser.io

Con *Phaser.io* podemos construir juego en HTML5, tanto para dispositivos móviles como para equipos sobremesa. Se trata de un *framework* de código libre bastante nuevo y en pleno crecimiento, proporcionando herramientas de rápido desarrollo que ayudan a manejar las tareas que completan el juego. Este *framework* está basado en *Pixi.js*, que se trata de una biblioteca con la que podemos crear imágenes y gráficos interactivos, juegos y aplicaciones sin necesidad de meternos de lleno en la API de *WebGL* y evitando problemas de compatibilidad con los navegadores.

Veamos unas capturas a modo de ejemplo sobre las interfaces de algunos de los juegos creados con *Phaser.io.*

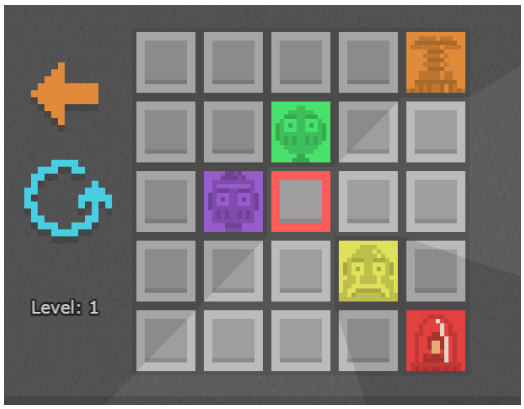

Imagen 4 - *Ejemplo Phaser.io Gravitron*

Gravitron es un juego 2D creado con *Phaser.io* cuya finalidad es encajar el cubo rojo en el cuadrado con los bordes rojos, con la dificultad de no poder parar hasta que se encuentra con otro cubo, una vez elegida la dirección del movimiento.

Imagen 5 - *Ejemplo Phaser.io Parallel*

Parallel es otro juego creado con *Phaser.io* con un desarrollo 2D que consiste en dirigir la bola hacia el final de la pantalla saltando obstáculos y combinando la parte superior con otra inferior para ello, utilizando los botones arriba y abajo para cambiar entre ellas y la barra espaciadora para saltar.

El funcionamiento de *Phaser.io* está dividido por estados que se basan en *callbacks*, (funciones integradas que nos ayudan con el desarrollo del juego), los principales son:

- ***Preload***: carga imágenes y varios recursos del juego al invocarlo.
- ***Create***: sirve para crear los personajes, el fondo, los enemigos, sonidos y animaciones.
- ***Update***: este estado está en continua ejecución dependiendo del tiempo que necesitemos, se encarga de configurar los movimientos de los personajes, así como colisiones e interacciones con el mundo.

Creado por *Unity Technologies*, *Unity* es un motor gráfico orientado a juegos multiplataforma programado en *C++, C Y Sharp* que permite elaborar juegos en 2D y 3D. Su soporte multiplataforma es líder hoy en día, pues permite conectarse con gran cantidad de los dispositivos existentes, plataformas móviles, escritorio, consolas, Web, TV y es compatible con la mayoría de los sistemas como Microsoft Windows, OS X, Linux y Android.

Pero no es solo un motor, *Unity* ofrece también servicios integrados como aplicaciones para manejar los ingresos de los usuarios, herramientas de análisis, sincronizar proyectos, compilar y compartir creaciones de forma automática.

Entre sus características cabe destacar su rapidez a la hora del desarrollo debido a su interfaz intuitiva que permite la creación de plataformas con renderizado de una manera totalmente personalizable y flexible sin la necesidad de tener una plantilla de programadores y artistas que se encarguen de llevar a cabo la construcción del juego. Tanto es así que los propios usuarios comparten sus códigos para poder ayudar los demás en sus problemas con el desarrollo.

Unity tiene un futuro prometedor dentro del mundo de los videojuegos ya que está muy en uso por desarrolladores independientes.

Conclusión

En nuestro proyecto hemos optado por *Phaser.io.* Hemos querido apostar por un motor gráfico nuevo y más sencillo que *Unity*. Nuestra idea inicial era hacer un juego ligero y en un lenguaje web, optamos por *JavaScript*, y *Phaser.io* es el que mejor se amoldaba a nuestros requisitos.

Comunicación con cliente y servidor

Para realizar las conexiones entre el cliente y el servidor necesitamos de una tecnología que sea capaz de realizar dicha tarea de una forma rápida y eficiente

Websockets

WebSocket es una tecnología que proporciona la conexión bidireccional por medio de un único *socket* entre el cliente y el servidor. Posee una API para el desarrollo con *Websockets* y su funcionamiento básico es el siguiente:

Primero debemos ejecutar el constructor para abrir una conexión *Websocket*.

```
var connection = new WebSocket('ws://html5rocks.websocket.org/echo',
['soap', 'xmpp']);
```

Imagen 6 - *Constructor Websocket*

Ponemos especial atención en el comando ws; significa que estamos estableciendo una nueva conexión *Websocket*. También hay wss, utilizado para conexiones seguras, al igual que *Hypertext Transfer Protocol Secure* (HTTPS) se usa para una conexión tipo HTTP segura. Este tipo de conexión también nos ofrece la posibilidad de añadir varios controladores de conexión para saber cuándo se mantiene abierta la conexión y el uso de subprotocolos y control de errores.

```
// When the connection is open, send some data to the server
connection.onopen = function () {
  connection.send('Ping'); // Send the message 'Ping' to the server
};

// Log errors
connection.onerror = function (error) {
  console.log('WebSocket Error ' + error);
};

// Log messages from the server
connection.onmessage = function (e) {
  console.log('Server: ' + e.data);
};
```

Imagen 7 - *Controladores y control de errores Websocket*

A continuación, tenemos que establecer conexión con el servidor:

```
// Sending String
connection.send('your message');

// Sending canvas ImageData as ArrayBuffer
var img = canvas_context.getImageData(0, 0, 400, 320);
var binary = new Uint8Array(img.data.length);
for (var i = 0; i < img.data.length; i++) {
  binary[i] = img.data[i];
}
connection.send(binary.buffer);

// Sending file as Blob
var file = document.querySelector('input[type="file"]').files[0];
connection.send(file);
```

Imagen 8 - *Websocket conexión con Servidor*

Esto se produce cuando se activa el evento *open*, entonces ya podemos enviar mensajes al servidor mediante el método *send*; así mismo, también el servidor puede enviarnos mensajes mediante la llamada *onmessage*.

La conexión tipo *Websocket* tiene una principal característica, y es que utiliza comunicaciones de origen cruzado, esto te permite la comunicación entre las partes de cualquier dominio, siendo el servidor quien decide si ofrece su servicio a todos los clientes o a un conjunto que esté dentro de un dominio. Pero esta tecnología tiene una desventaja respecto a las otras, y es que es incompatible con los servidores proxy que utilizan las conexiones HTTP.

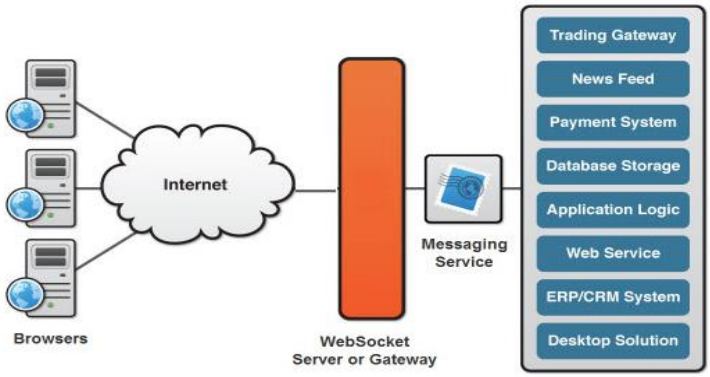

Imagen 9 - *Servidor Websocket*

Este tipo de tecnología, se utiliza principalmente para la creación de videojuegos online multijugador, chats, actividades online e información deportiva. Como podemos observar, *Websocket* se utiliza siempre que exista la necesidad de crear una conexión en tiempo real y con una baja latencia entre cliente y servidor.

Socket.io

Socket.io nos permite establecer una comunicación bidireccional para aplicaciones web en tiempo real mediante eventos. Una de las ventajas más importantes de usar *Socket.io* es que no nos tenemos que preocupar por la compatibilidad entre navegadores.

Para las aplicaciones web, es muy importante la retroalimentación entre los usuarios, por lo que comenzaron a usarse los *sockets TCP (Transmission Control Protocol)* entre desarrolladores para enviar y recibir datos. La primera versión de *Socket.io* vio la luz el día 28 de mayo de 2014, incluyendo una API de alto nivel permitiendo crear nuevas aplicaciones dándole uso.

La forma de comunicación cliente y servidor entre los *sockets* se realiza a través de señales. El cliente emite una señal, el servidor tiene un manejador que se encarga de buscar la respuesta para la misma, y éste emite de nuevo otra señal que interpretará con su manejador el cliente. Veamos un ejemplo:

```
var server = require("net").createServer();
var io = require("socket.io")(server);

var handleClient = function (socket) {
    // we've got a client connection
    socket.emit("tweet", {user: "nodesource", text: "Hello, world!"});
};

io.on("connection", handleClient);

server.listen(8080);
```

Imagen 10 - *Ejemplo Instancia Servidor*

```
socket.on("connect", function () {
    console.log("Connected!");
});
```

Imagen 11 - *Ejemplo Instancia Cliente*

Conclusiones

De entre las tecnologías estudiados, hemos escogido *Socket.io*. Es una implementación de *Websockets* muy avanzada, entre sus principales funcionalidades encontramos que utiliza diversas tecnologías para adaptarse al navegador utilizado, permite las reconexiones del cliente, está actualizándose constantemente, y el uso de canales (*rooms*) necesario en nuestro proyecto, es de los más avanzados hasta la fecha.

Herramientas de desarrollo

A continuación, se nombran las herramientas de apoyo que hemos utilizado para el desarrollo del código de este proyecto.

Google Chrome

Google Chrome es considerado el navegador más rápido del mundo, creado por la compañía Google INC en septiembre de 2008, ya es el navegador líder del mercado. Es compatible con cualquier sistema operativo, y está disponible en alrededor de unos 50 idiomas. Su principal ventaja es la agilidad a la hora de interpretar código *JavaScript* que utiliza la mayoría de las páginas web.

Hace unos años ningún navegador disponía de herramientas adecuadas para el desarrollo de aplicaciones web, por lo que dependías de herramientas de terceros, que no siempre eran compatibles con el navegador actual o no se integraban correctamente. Ahora esto ha cambiado y los navegadores incluyen herramientas, como la consola y el depurador de código. Nosotros nos vamos a centrar en la consola de Google Chrome.

Una de las partes más importantes para los desarrolladores es la depuración de código, y la búsqueda y corrección de errores por lo que la consola de Google Chrome es necesaria para apoyar nuestro proyecto.

Utilizamos la consola de depuración en nuestro proyecto, ya que nos proporciona muchas de las herramientas que necesitamos para comprobar el correcto funcionamiento del código de nuestro juego. Esta consola nos permite ver los mensajes de información, error o alerta que recibimos al realizar la carga de los elementos seleccionados; incluye depurador de código y también nos permite interactuar con la página mediante comandos *JavaScript*.

La consola contiene todas las variables y objetos declarados en el código de la página, por lo que podemos acceder a ellas mediante llamadas. En cuanto a la depuración del código podemos elegir el archivo sobre el cual queremos ejercer la depuración, nos ofrece también la posibilidad de añadir *breakpoints* sobre la línea interesada y añadir un observador para poder ver lo que almacenan las variables que nos interesen.

Git

A la hora de trabajar y debido a la distancia de cada uno de los miembros del grupo, usar una aplicación que controle las versiones y nos facilite la información sobre cada cambio es fundamental.

Existen varios *softwares* de control de versiones como SVN, Git, ACCUREV y varios más. Nosotros vamos a utilizar Git, porque reúne las condiciones necesarias para nuestro proyecto. Podemos controlar quién ha modificado cada elemento y cuando lo ha hecho, podemos observar cómo ha cambiado el proyecto con el tiempo, con la ayuda de las etiquetas podemos controlar sus versiones y poder retomar alguna antigua por si no nos satisface el resultado. Es una aplicación que mejora la capacidad del trabajo en equipo.

Sublime

Sublime es un editor de texto multiplataforma que posee soporte para casi todos los lenguajes de programación añadiendo colores para las variables y etiquetas específicas de cada lenguaje.

Está pensado para programar; tiene una interfaz oscura para no fatigar los ojos, y se trata de uno de los editores de texto más utilizados para este fin, el desarrollo.

También destacar que, gracias a la API en *Python* de *Sublime*, se pueden desarrollar *plugins* para expandir el editor de texto y aumentar sus funcionalidades, existen *plugins* muy útiles como el que permite integrar Git en *Sublime* y hacer *commits* desde el.

Capítulo 4: Descripción del Sistema

En este apartado se va describir la aplicación y todas sus funcionalidades, junto a una descripción de requisitos mínimos tanto para su uso como cliente como su ejecución en un servidor.

Ámbito del sistema

El objetivo del proyecto es desarrollar un juego y un *bot* en la multiplataforma de mensajería Telegram. Se trata de una aplicación totalmente online, que aprovecha la plataforma y el *bot* como lanzador y proveedor de información del usuario al juego.

Es el propio juego quien se encarga de gestionar la información recopilada y de crear conexiones personalizadas a cada usuario para la transferencia inmediata de información entre los jugadores conectados a la aplicación.

Funcionalidades de la Aplicación

La aplicación dispone de dos partes, el *bot* en Telegram y el propio juego.

El primero de ellos, es el que se encarga de obtener la información del usuario. Cuando se ejecuta el *bot*, éste busca los datos del usuario que está interactuando con él, datos tales como el nombre del usuario o su identificador en Telegram. Una vez obtenidos estos datos, el *bot* queda a la espera de que el usuario le dé la orden de jugar. Cuando recibe esa orden, el *bot* genera un enlace al juego y lo muestra en el chat. Basta con pulsar el botón de jugar para que el *bot* abra una ventana de navegación con el juego y le mande toda la información del usuario para crear una conexión personal.

El juego, con la conexión ya generada, carga los componentes necesarios para arrancar y le muestra al usuario el menú con el cual ya puede interactuar.

El *bot* **Track and Field** se puede invocar en conversaciones grupales, individuales o hablando con el directamente. Se puede buscar en el buscador de contactos o invocar en las conversaciones con el comando `@gametfbot`

Exploración de la Aplicación

A continuación, se va a mostrar un caso de uso del *bot*, para abordar sus funcionalidades y explicar cómo es la interacción con el juego, tanto en una conversación grupal como en una interacción directa con el *bot*.

Para comenzar, se va a buscar al *bot* **Track and Field** en el buscador de Telegram, como si de un usuario de la aplicación de mensajería se tratase; lo buscamos por su nombre `@gametfbot`.

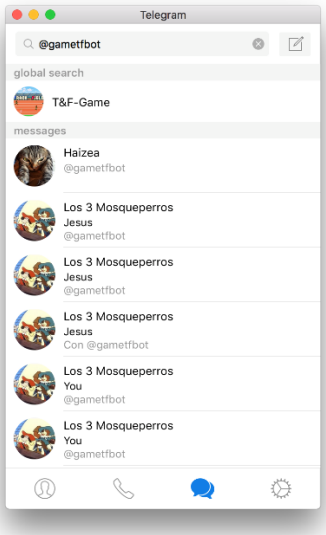

Imagen 12 - *Búsqueda Track & Field*

Una vez encontrado, abrimos una conversación con él. En dicha conversación el *bot* va guiando al usuario para que pueda ejecutar de forma sencilla el juego.

En su primera ejecución al usuario le aparecerá un botón *Start*, este botón iniciará el *bot*, una vez iniciado el usuario el usuario recibirá el saludo personalizado del *bot* y le dirá que escribiendo el comando `/play` o pinchando en el empezará a jugar.

Tras ejecutar el comando `/play`, en el chat aparecerán los datos del usuario y un enlace al juego. Dando a jugar se abrirá una nueva ventana (dentro de Telegram o en el navegador, según dispositivo y plataforma) con el juego.

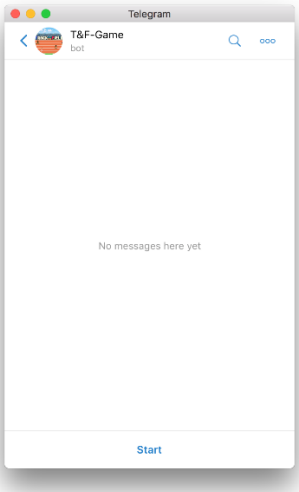

Imagen 13 - *Start bot Track & Field*

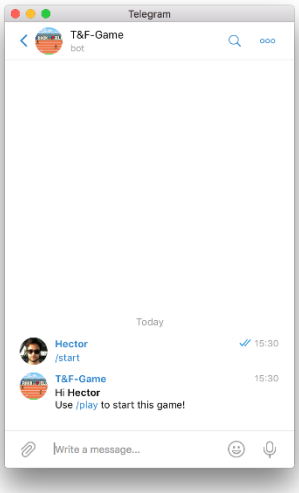

Imagen 14 - *Play bot Track & Field*

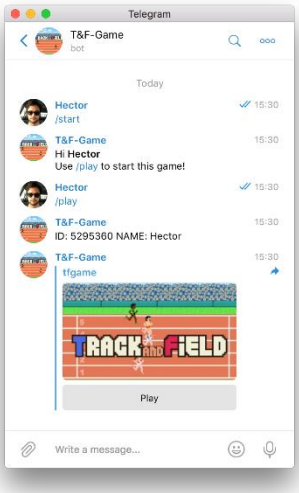

Imagen 15 - *Start bot Track & Field*

A continuación, se va a mostrar la invocación del bot en una conversación, en este caso grupal, aunque el funcionamiento es el mismo en una conversación privada. Es una interacción más rápida y sencilla que en el caso anterior, ya que la interacción entre amigos es el público objetivo de este juego.

Para comenzar, abrimos una conversación en Telegram, y en el chat escribimos el comando del *bot*, `@gametfbot`.

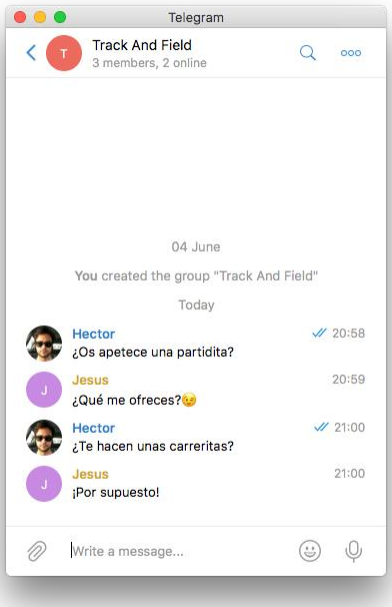

Imagen 16 – *Chat de grupo*

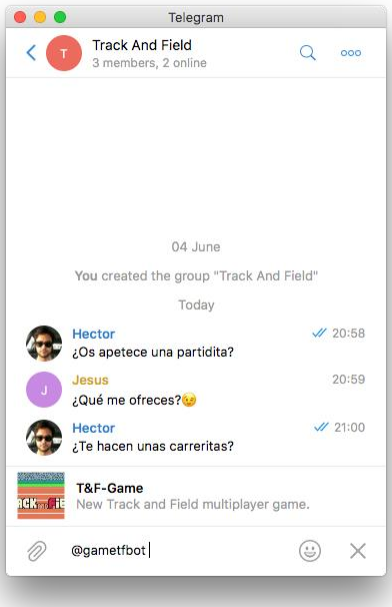

Imagen 17 - *Llamada Track & Field en grupo*

Una vez escrito el comando y sin pulsar nada nos aparece encima del cuadro de texto el nombre del juego **T&F-Game**, le seleccionamos y compartirá el enlace del juego en la conversación para que el resto de usuarios puedan entrar.
Con el juego ya compartido, solo queda jugar.

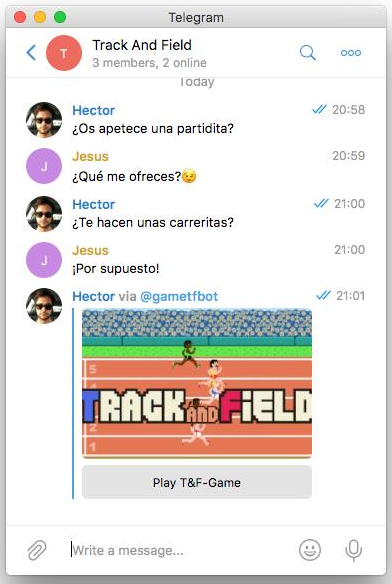

Imagen 18 – *Track & Field compartido en grupo*

Vamos a pasar a la ejecución del juego, y empezamos por el menú principal, que es la pantalla inicial y lo que primero ven los usuarios.

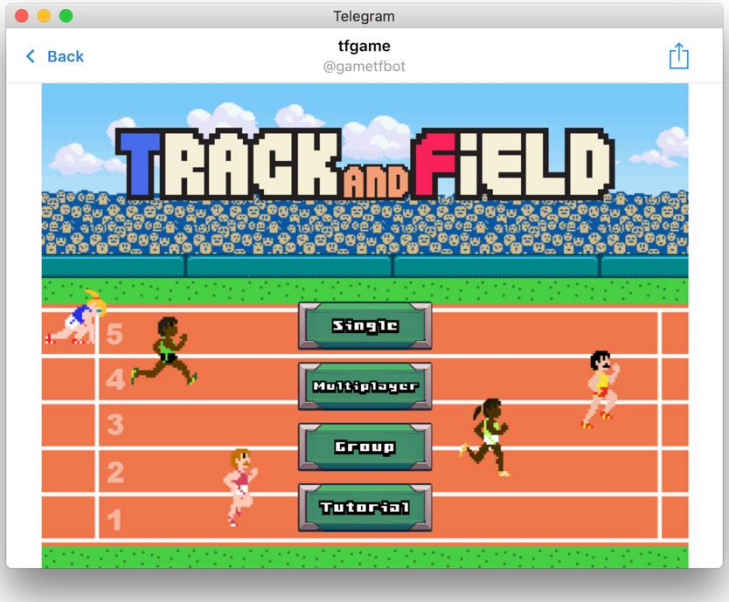

Imagen 19 - *Menu Track & Field*

Track and Field se compone de tres modos de juego: el modo *Single* u *offline*, el modo *Multiplayer* u *online* y el modo *Group* o grupo.

A demás de estos modos de juego que se explican más adelante, nos encontramos el tutorial, que explica la mecánica del juego y en qué consisten los distintos modos.

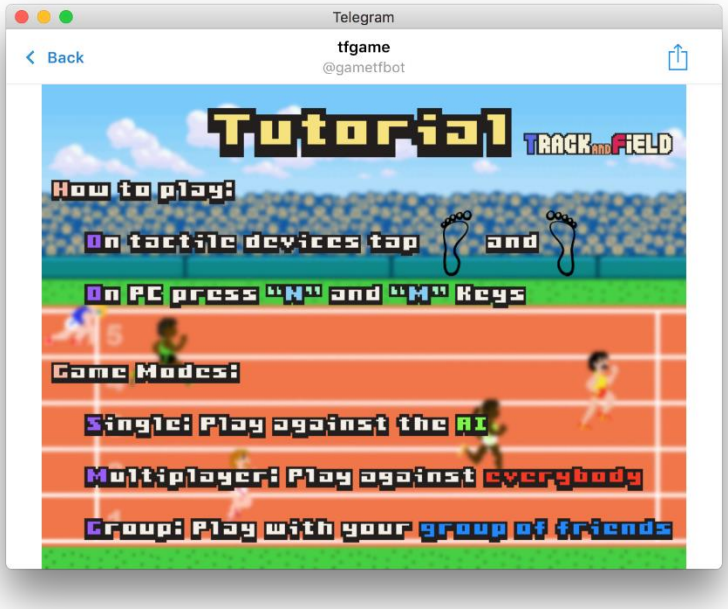

Imagen 20 - *Tutorial Track & Field*

Una vez visto el tutorial, nos quedan los tres modos de juego. Vamos a empezar por el modo *Single* o modo offline. En este modo, el usuario jugará solo contra la máquina, para ello a su partida se le añadirán cuatro personajes (llamados BOT_X) controlados por la Inteligencia Artificial básica del juego.

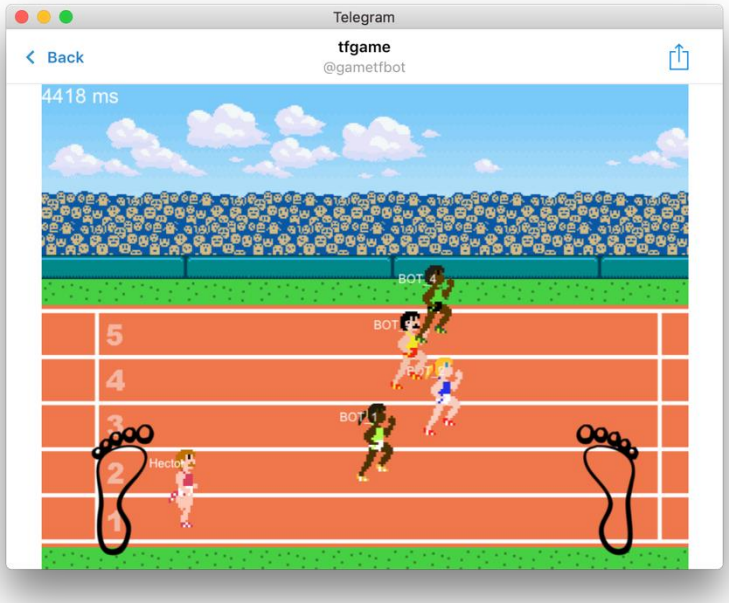

Imagen 21 - *Single Track & Field*

Después del modo *Single,* nos encontramos el modo *Multiplayer* u *online*, en este modo el jugador se enfrenta a otros jugadores conectados en la aplicación, desde cualquier parte del mundo.

Al entrar en el modo *Multiplayer,* el jugador queda en un *lobby* o sala de espera, aguardando a que se conecten otros cuatro jugadores más y se inicie la partida. Según se van creando partidas, se van generando nuevos *lobbies*, por lo que no hay una única partida multijugador en el juego, sino que hay múltiples partidas ejecutándose a la vez. De esta forma se puede dar servicio a una gran cantidad de usuarios y permite que el juego sea dinámico y rápido.

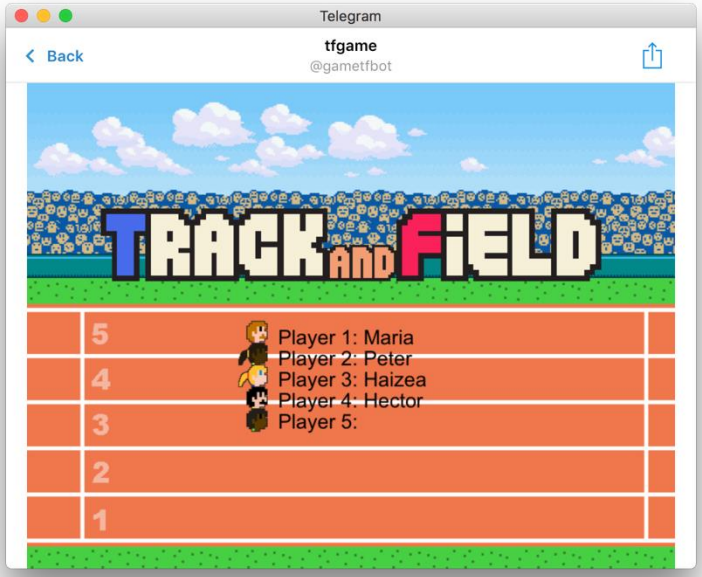

Imagen 22 - *Multiplayer lobby Track & Field*

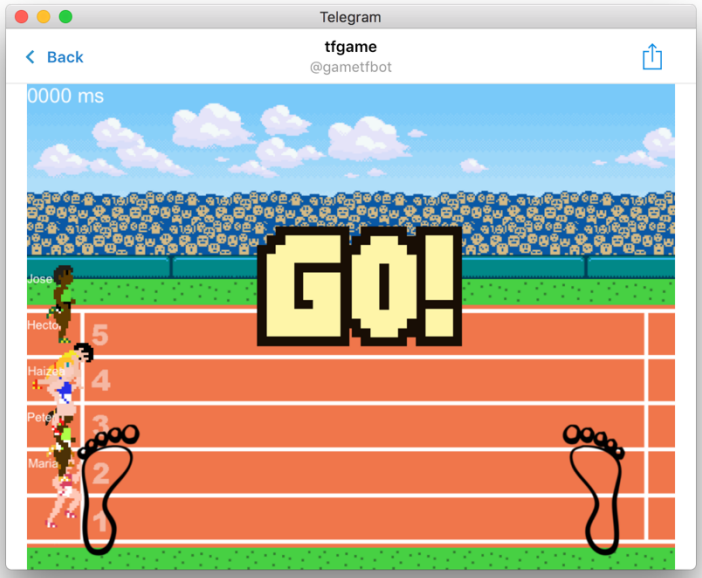

Imagen 23 - *Multiplayer partida Track & Field*

Al finalizar la partida, a cada usuario se le muestra su puntuación y el puesto en el que ha quedado, indicándole si ha ganado al resto o no.

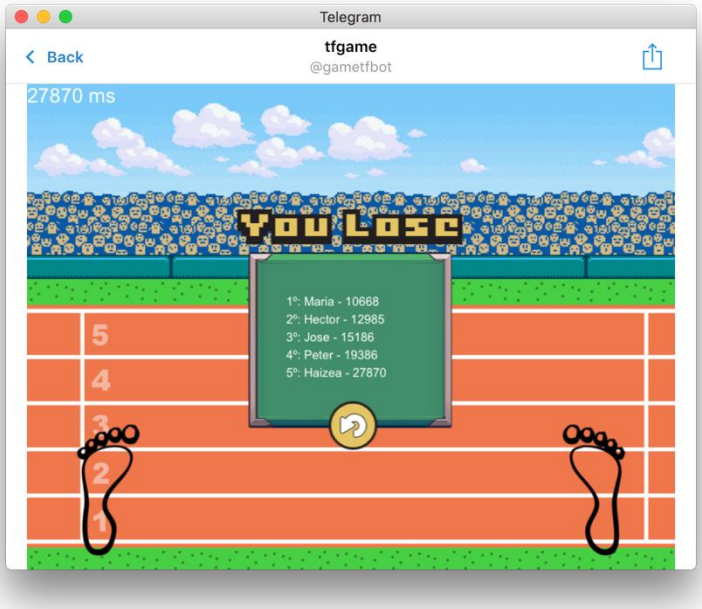

Imagen 24 - *Resultados Multiplayer Track & Field*

Por último, el modo *Group* o grupo, en esta modalidad el jugador se enfrenta solo a las personas pertenecientes a la conversación en la que se ha invocado al bot.

Al entrar, los jugadores esperan en el mismo y único *lobby* por grupo a que se conecte el resto de miembros. En este modo la partida no se inicia solo al llenar la sala, sino que también puede empezar manualmente pulsando el botón *Start*, por si el grupo es de menos de cinco integrantes o solo quieren jugar unos pocos. Cualquier miembro del grupo que esté en la sala puede darle a comenzar la carrera.

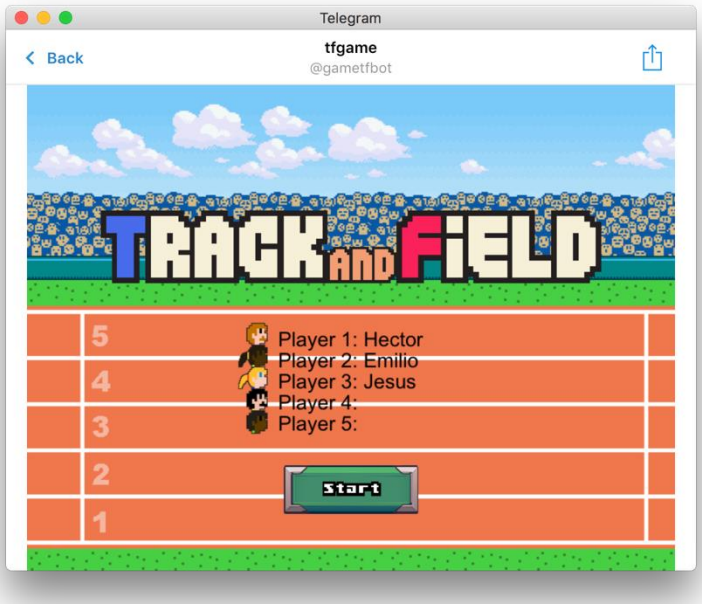

Imagen 25 - *Group lobby Track & Field*

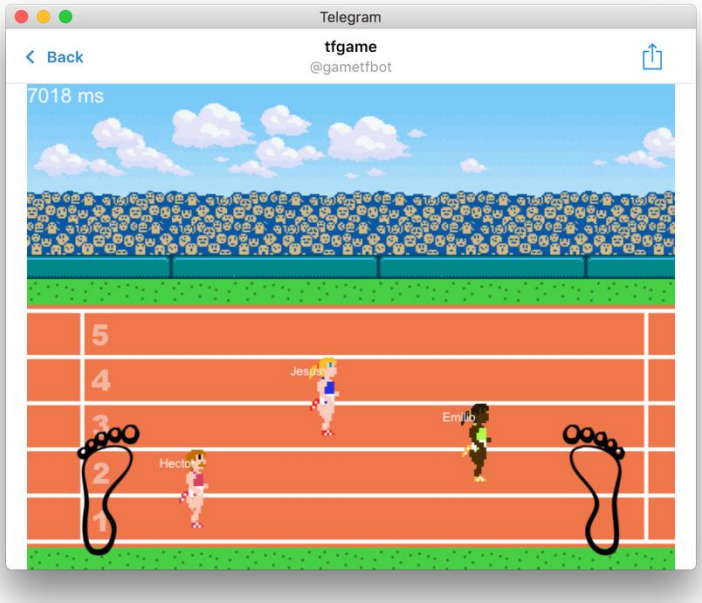

Imagen 26 - *Group partida Track & Field*

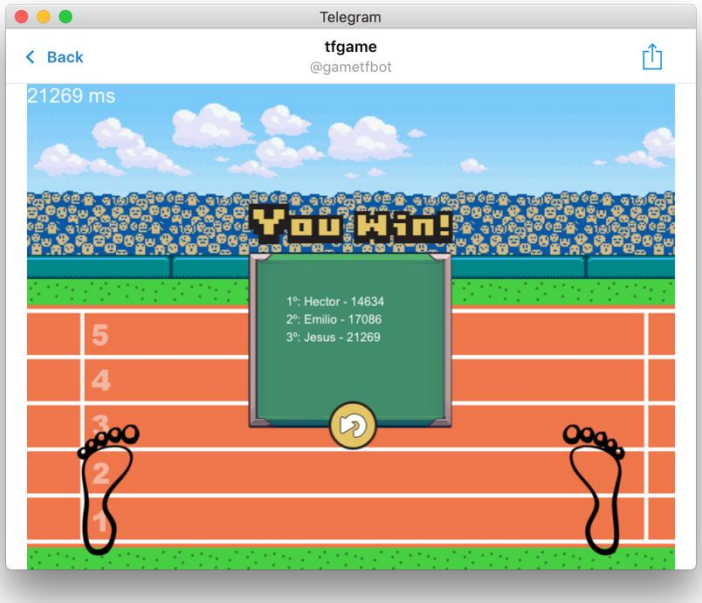

Imagen 27 - *Resultados Group Track & Field*

El juego posee una jugabilidad sencilla, ya que el usuario solo tiene que tocar dos botones más rápidos que el resto para ganar. Pero lo importante no era la complejidad del juego en sí, sino la correcta y optima gestión de las conexiones por parte del servidor y el cliente y la integración con Telegram.

Requisitos

En este apartado se va a hablar de los requisitos necesarios por parte del usuario o cliente, para que pueda ejecutar la aplicación, y por parte del servidor, para poder instalarlo.

Requisitos por parte del cliente

Siendo Telegram una aplicación multiplataforma era necesario que **Track and Field** también lo fuese. Los requisitos son los mismos para todas las plataformas, siendo:

- Disponer de un usuario de Telegram.
- Aplicación de Telegram actualizada a la última versión.
- Navegador actualizado y habilitada la opción de ejecutar código *JavaScript*.
- Conexión a internet estable.

Requisitos por parte del servidor

Tanto el *web service* del *bot* en Telegram como el del cliente-servidor, están desarrollados con *Node.js*. Tenerlo instalado es el único requisito para ejecutar e instalar los servidores.

Capítulo 5: Arquitectura del sistema

Una vez especificados los requisitos, se desarrolla un diagrama que servirá de resumen y a la vez de base de la aplicación con el fin de poder continuar su desarrollo de forma sencilla y orientada. A continuación, se explicará paso a paso y de forma concreta el funcionamiento del sistema.

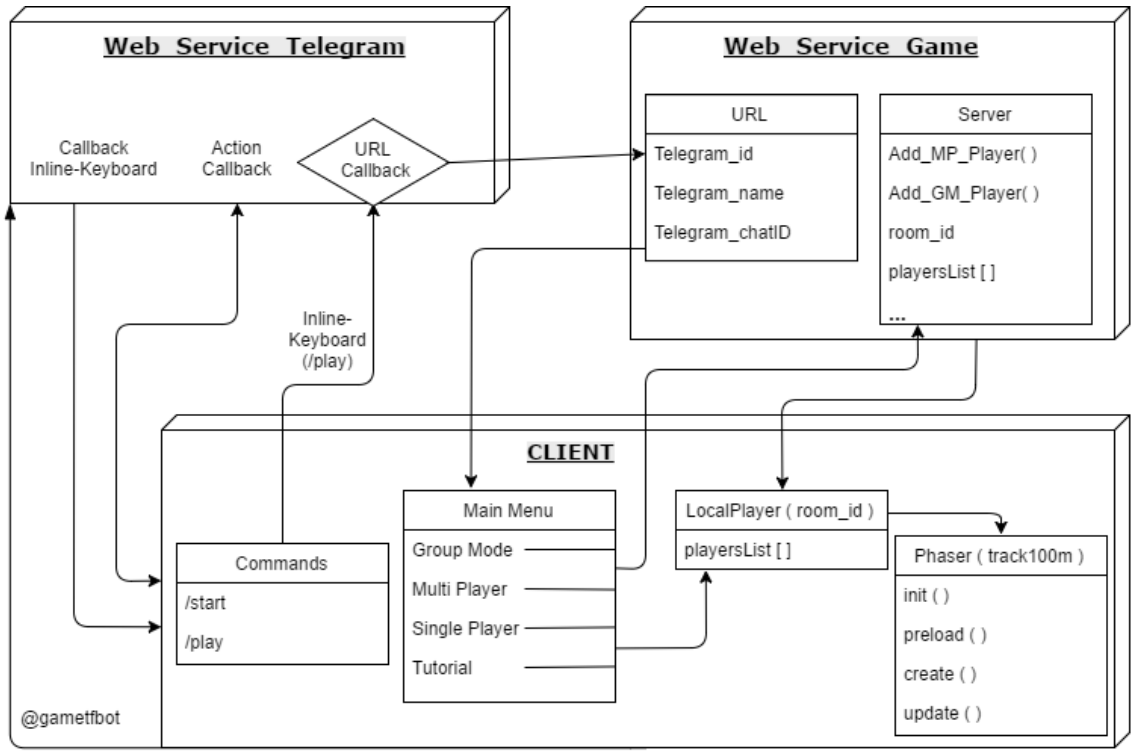

Imagen 28 - *Arquitectura del sistema*

Web service para Telegram

Para comenzar, se ha instalado un *web service* en la universidad que nos permite la comunicación entre el cliente y el servidor a tiempo real. Se utiliza como túnel entre Telegram y la aplicación, así los usuarios, añadiendo el *bot* en Telegram, son capaces de comunicarse fácilmente con él por medio de comandos; el principal comando es `/play`, que se usará como lanzadera. Con este comando, el usuario recibirá la opción de jugar, y una vez haga *click*, la misma API de Telegram, implementada en el *web service*, responderá en un *callback* con la URL (*Uniform Resource Locator*) del juego, la cual será lanzada desde la propia aplicación de mensajería.

Para añadir el bot será necesario escribir `@gametfbot` en cualquier instancia de chat de Telegram (ya sea de grupo o privado), una vez llamado ofrecerá la opción de juego a todos los usuarios presentes en la sala.

Web service para el juego

Una vez se hace *click* en jugar, Telegram lanza una ventana de navegación con el contenido de la URL obtenida del *callback*. En este momento el nuevo *web service* se encarga de recopilar los datos de la URL (la ID de Telegram, el nombre utilizado por el usuario y la ID del chat en cuestión).

Estos datos son recibidos por el cliente que ha hecho *click* previamente, y aparece la primera vista del juego, el menú principal, con el cual el usuario podrá interactuar. Arriba en el diagrama, aparecen los distintos modos de juego. En este momento, ya podemos decir que el usuario está identificado.

Primero hemos invocado al *bot* de Telegram, el cual nos ha dado la opción de lanzar el juego. Al lanzar el juego, se comparte la información del usuario de Telegram entre *web services* por medio de una URL. El servidor del juego identifica la URL y genera información del jugador a través de los datos recopilados con Telegram.

A continuación, vamos a explicar el funcionamiento paso a paso de cada uno de los modos de juegos una vez capturada la información y enviada al servidor del juego:

- **Tutorial:** Se realiza en el lado del cliente, por lo que no necesita comunicarse con el servidor de nuevo, cambia la vista mostrando unas instrucciones básicas para jugar.

- **Single Player:** Es el modo de práctica. El jugador local compite contra cuatro jugadores controlados por la máquina. No necesita comunicación con el servidor, ya que la información propia del jugador es suficiente. Los cuatro jugadores son generados automáticamente.

- **Multiplayer:** Una vez entramos a este modo, se emite información al servidor por medio de *sockets*. Se añade un nuevo jugador en la lista en el lado del servidor (el jugador que invoca), el servidor comprueba la lista, y siempre que sea menor que cinco (límite de jugadores en la pista en este modo), éste pertenecerá a la misma *room (sala de jugadores)*. Una *room* dentro de los *sockets*, es una forma de limitar la comunicación cliente-servidor entre un grupo de usuarios, por lo tanto, los primeros cinco jugadores compartirán un identificador de *room* común y único, los siguientes cinco otro, etc. Así aseguramos que no haya interferencias en la comunicación entre *sockets*, aislando la información del exterior.

  Una vez el servidor añade a la lista el jugador, y le asigna la *roomID*, la lista completa es devuelta al emisor inicial y a los demás usuarios de la misma. Por lo tanto, cada jugador que entre en el modo, irá compartiendo la lista de jugadores entre sí mientras que compartan la misma *room*, por lo que por cada *room* tendremos cinco jugadores que tendrán la lista de los mismos de forma local.

Una vez la sala de espera se llena (*matchmaking*), la instancia local de cada jugador (del lado del cliente) lanza la nueva vista, que será la pista donde correrán. Cada jugador tiene asignada la calle en la que realizará la carrera y el *sprite* (modelo gráfico del jugador) propio. Por cada instancia local, se tiene una lista de cuatro enemigos, que nos permite actualizar la vista cada vez que cambien de velocidad o posición.

- **Group Mode**: Es muy similar al modo anterior, pero orientado a los grupos de Telegram. La idea es jugar online, pero con conocidos, con tus amigos de cualquier grupo de Telegram. Una vez invocado el *bot* desde un grupo, cualquier usuario del mismo podrá hacer *click* en jugar y unirse a la partida en el *Group Mode*. Como hemos explicado anteriormente, se envía información a través de la URL, siendo una de ellas el *ChatID*. Éste es un identificador único común a todos los miembros del grupo, por lo tanto, si hay veinte usuarios en un grupo, los veinte compartirán el mismo *ChatID*, del mismo modo que en un chat privado, los dos usuarios compartirán el mismo identificador. Por lo tanto, es muy útil para relacionarlos y poder así jugar en grupo.

  Al entrar, los usuarios en este modo realizan la misma funcionalidad que en el *Multiplayer*, pero en vez de generar *room* dinámicamente, como hacemos en el otro modo, la ID de la *room* será el *ChatID*; como es evidente todos los miembros de ese grupo compartirán sala de espera.

  Otra diferencia respecto del modo anterior, es la posibilidad de jugar sin esperar el llenado de la sala (cinco personas). Esto quiere decir que, si estás en un chat privado, o en un grupo de tres, también podrás competir con ellos al instante. Una vez estén en la lista todos los jugadores deseados, pulsando el botón *Play Now* la lista es enviada al siguiente estado (cambio de vista) y se inicia la carrera entre los miembros que hubiese en el momento de darle al botón.

Estados de la aplicación en el cliente

Gracias al motor gráfico (*Phaser.io*), hemos dividido el proyecto en varios estados, facilitando el trabajo al ser modular, y permitiendo cambios futuros de forma sencilla simplemente añadiendo un nuevo módulo, sin tener que modificar otros.

A continuación, se explican los distintos módulos de la aplicación:

- **index.html**: Es la puerta a la aplicación, la ventana contenedora de las vistas, y donde están vinculados cada uno de los archivos necesarios o incluidos. Incluye todos los archivos *JavaScript* necesarios, librerías (*Phaser.io, Socket.io*…), modelos y estados del juego.

- **game.js**: Se inicia la aplicación, dentro de la misma encontramos la inicialización de la ventana del juego. Después, se realiza la conexión del cliente con el servidor mediante *socket*; si todo ha ido correctamente, se recibe la información recogida

de la URL desde el servidor. Por último, se cargan los estados posibles del juego dentro de la aplicación y se lanza el primero de ellos (*boot*).

- **boot.js**: Se escala la pantalla y se carga la barra de carga que será utilizada en el siguiente estado, el cual es invocado.

- **preload.js**: Mediante la barra de carga anterior (la cual se mantiene en todo este estado), se cargan todas las imágenes, sonidos y demás útiles de la aplicación, que serán almacenados en memoria y enviados de estado a estado, evitando que vuelvan a tener que ser cargados.

- **menu_modes.js**: Vista sencilla del menú del juego mencionado anteriormente con los distintos modos, *single player, multiplayer*, etc. Según el botón pulsado variará el estado siguiente.

- **single_player.js**: Utiliza dos funciones del motor gráfico, *create()* y *update()*, la primera inicializa al jugador local leyendo los datos del servidor (*socket*) y lo guarda en la lista de jugadores; la segunda se encarga de esperar hasta que el jugador es cargado, y una vez en la lista, se rellena con otros cuatro jugadores (controlados por la máquina en este caso) y se lanza el estado de la carrera (*track100m*).

- **tutorial.js**: Carga una imagen sencilla y explicativa del juego; una vez se hace *click* en ella desaparece y aparece nuevamente el menú principal.

- **multi_player.js y group_mode.js**: Ambos utilizan la función *create()* para inicializar los datos del jugador local. Una vez iniciados, son enviados al servidor a través del socket y el servidor devuelve la lista actualizada de jugadores (incluido nuestro jugador). Una vez se cumplen los requisitos, bien cinco jugadores en la sala de espera, o en el caso del *group_mode*, que alguien pulse el botón de *Start*, la función *update()* entra en acción, y llama al estado de la carrera (*track100m*).

- **track100m**: Se encarga de la lógica del juego. Primero, en la función *init()* se cargan los atributos pasados desde los estados anteriores, como es el caso de la lista de jugadores. En la función *preload()* se asignan atributos para evitar que con el cambio de pantalla se pause el juego. La función *create()* inicializa todos los estados de la partida, temporizadores, sonidos, la pista en sí con los límites de la misma, y por supuesto, los controles (tanto para PC como para dispositivos con pantalla táctil).

  Usando la lista completa de jugadores recibida del estado anterior, dando igual el modo de juego del que proceda, buscamos a nuestro jugador en la lista (el que

coincida con el identificador único del *socketID* de la instancia local) y añadimos los demás enemigos de la lista como enemigos en otro *array*.

Cada jugador en su propia instancia local tendrá la lista de todos los jugadores (incluido el mismo) y a los otros cuatro como enemigos en otra lista. La razón por la que dividimos la lista inicial con todos los jugadores en jugador local y enemigos, es que el jugador local se mueve con la combinación de botones, mientras que los enemigos necesitan ser actualizados ya que sus estados (posición/velocidad) no dependen de la instancia local (salvo en el modo *single player* con los *bots*) sino del estado que recibe desde cada uno de los enemigos.

Para terminar, la función *update()* se encarga de comprobar todas las variables inicializadas anteriormente, el tiempo, la velocidad, la posición de los enemigos, si el jugador local se está moviendo (por medio de la combinación de teclas), si todos han llegado a meta, y por supuesto, de mostrar los resultados al final de la partida. Una vez terminada la partida, se reinician los estados.

Estados de la aplicación en el servidor

En este apartado vamos a hablar del **Server.js**. Es el *web service* del juego mediante el cual los usuarios se conectan pudiendo formar *rooms* para jugar por grupos o en multijugador. Para empezar, se encarga de recolectar la información de la URL, obteniendo así el id/nombre del usuario en Telegram y el *chatID*, que será el identificador de las *room* en el modo *group*, y enviarla a cada cliente que la pida. Una vez se solicita conexión (al entrar al juego), el servidor emite por medio de *sockets* la información, y el usuario queda identificado por su *socketID*. El servidor dispone de un manejador de eventos, encargado de ejecutar una función u otra, dependiendo de la orden emitida por el cliente en el *socket*.

También se encarga de gestionar las *room*, haciendo posible el sistema de *matchmaking* a tiempo real. Cada vez que un cliente entra en el modo *group* o *multiplayer*, éste es añadido al *array* de jugadores del servidor identificado por la *room* en la que esté, y desde ese momento, solo los mensajes referidos a la *roomID* asignada llegarán al destinatario. Por supuesto, también se encarga de las desconexiones, eliminando al usuario de la lista que corresponde, y avisando a los demás clientes de la *room* del suceso de forma inmediata.

Estados de la aplicación en el conector

El conector es un *web service* que nos permite la entrada/salida de datos, en este caso utiliza una implementación de la API de Telegram basada en *JavaScript* que se utiliza como lanzadera. El cliente llama al *web service* mediante una orden, escribiendo `@gametfbot` en cualquier chat de Telegram, invocando al *bot* directamente añadiéndolo como un contacto más, o desde cualquier navegador accediendo a la siguiente URL: “https://telegram.me/gametfbot”.

Telegram utiliza un *token,* que es un identificador único para el *bot*; éste servirá para vincular la API de Telegram directamente con nuestro *bot*.

Como cualquier *bot* de Telegram, permanece a la espera mediante unos manejadores, y en cuanto recibe una petición de un usuario, comienza su funcionamiento. Se ejecuta automáticamente la orden `/start` que saluda al usuario y le invita a jugar. Si el usuario utiliza el comando `/play`, aparecerá una imagen con un botón de jugar, que, si es pulsado, comenzará el proceso de vinculación con el *web service* del juego, el cual será lanzado dentro de la misma aplicación de Telegram.

Capítulo 6: Problemas y soluciones

A continuación, se exponen algunos de los problemas que han aparecido en el proyecto y como se han solucionado.

Nada de información sobre juegos online en Telegram y el conector

Con la aparición de los *bots* para juegos en Telegram a finales de 2016, surgió bastante documentación online, pero no la suficiente, ya que al final una librería o una API se sostiene a base de una comunidad, y el problema comienza cuando vemos que apenas asciende a 20 el catálogo de juegos disponibles, y muchos menos, *mutliplayer online* como es el caso, por no decir ninguno.

Para empezar, se recopiló información sobre una nueva API en *JavaScript* en vez de *Python,* que es la que se había utilizado hasta el momento, ya que ésta última no disponía de la actualización para los juegos. Una vez elegida (solo había una con opción para juegos), se preparó el código base para permitir interaccionar con el *bot*, y más tarde se añadió la funcionalidad definitiva, permitiendo lanzar juegos en HTML5 desde Telegram.

Problema con el GET de información a la URL

Puede parecer sencillo para cualquiera que haya realizado una página web, pero la verdad es que no es así, al utilizar *Node.js* como *web service*, es necesario utilizar librerías externas (o al menos recomendable) para manejar eventos de *Request* y *Response*. Tras buscar mucho por la red, nos dimos cuenta que la opción más sencilla era usar *Express* que nos permitía realizar estas operaciones. Hasta aquí bien, obteníamos los datos de la URL, el problema es que es necesario hacer un *Response* ya que, sino el *web service* se queda a la espera, la solución es responder con el envío del *index.html* con la ruta relativa.

```
51  app.get('/', function(req, res) {
52    playerID = req.query.playerID;
53    playerName = req.query.playerName;
54    chatID = req.query.chatID;
55    res.sendFile(path.join(__dirname,'/../index.html'));
56  });
```

Imagen 29 - *Problema con el GET de información a la URL*

Problema con el uso de *sockets (rooms)*

Al realizar el juego modo *multplayer,* y no compartir todo un espacio común entre instancias, aparece el siguiente problema: cuando formamos una carrera con cinco clientes no hay problema ya que toda la información que se comparta entre los mismos y el servidor, y las respuestas que éste emita a todos (movimientos, conexiones, desconexiones…) serán correctas, pero cuando entran los siguientes, comienzan los problemas, la lista de jugadores cambia, y no podemos mantener la anterior. Es por esto que incluimos las *rooms* como solución. Por cada grupo de clientes asignamos una *roomID* única que nos permite desde el servidor seleccionar que mensajes y a que grupo de clientes va, quedando solucionado el problema del *multiplayer* por salas.

Problema al pasar datos al socket sin sobreponer las sesiones previas

Este problema surge al abrir dos instancias del juego, y estando con ambas en el menú. El servidor recogía la misma información para ambos, compartían nombre, por ejemplo. Se solucionó generando el *socket* nada más ejecutar la aplicación, y realizando un método que retornaba el *socket* del cliente en cualquier momento que fuese necesario. A su vez, desde el servidor se envía la información solo hacia el *socketID* del cliente que hace la petición, evitando así compartir esa información. Es el mismo sistema utilizado con las *room*; al generar una conexión se crea una *room* con el *socketID* del cliente automáticamente, por lo tanto, al enviar los datos hacia esa *room*, llega solo a un usuario.

```
70  // New socket connection
71  function onSocketConnection (client)
72  {
73    //If we cannot catch name/id from telegram, disconnect.
74    if(playerID == 0 || playerID == undefined || playerID == null)
75      onClientDisconnect();
76    else
77      //Send the telegram info to a specific client
78      socket.in(client.id).emit('playerInfo', {id: client.id, id_teleg: playerID,name: playerName, chat_id: chatID})
```

Imagen 30 - *Problema al pasar datos al socket sin machacar las sesiones previas*

Problema con el *socketID*

En un principio se intentó reutilizar el ID que genera la API de Telegram por cada usuario asignándola directamente al *socketID*. Esto supuso varios fallos e inestabilidad en las conexiones cliente/servidor, por lo que tuvimos que anularlo. Más adelante encontramos información útil sobre los *sockets*, en la cual aparecía que la librería se retroalimenta con ese *socketID*; si se modifica, genera múltiples problemas por dependencias.

Problemas con el *testing* de juegos *online*

A la hora de probar cada uno de los cambios que se han ido realizando, nos dimos cuenta que era muy tedioso en los modos multijugador, ya que necesitábamos lanzar varias instancias para testear, se solucionó probando con menos usuarios de los definitivos (dos por partida en vez de cinco) y más adelante se adaptó el código. También realizamos un *script* en *Shell* llamado *LAUNCHER.sh* que nos sirvió de gran ayuda para matar el proceso y volver a lanzar ambos *web services* (el conector o *bot* de Telegram y el juego en sí) con un solo comando.

```
1  #!/bin/bash
2  killall node
3  rm -f BOT_LOG.out
4  rm -f GAME/GAME_LOG.out
5  nohup ./launchBot.sh >> BOT_LOG.out&
6  nohup ./GAME/launchGame.sh >> GAME/GAME_LOG.out&
```

Imagen 31 - *Problemas con el testing de juegos online*

Problemas con la caché del navegador

Este error quizá sea el que más tiempo nos ha hecho perder, ya que desde el principio hemos sufrido diversos fallos ajenos a nuestro código. La primera ejecución de la aplicación siempre fue correcta, pero una vez que modificas y vuelves a lanzar todo, el resultado sigue siendo el mismo, o incluso aparecen errores gráficos o desincronizaciones.

La solución en Windows, utilizando el navegador Chrome, es o bien cerrarlo por completo y volver a abrirlo, o usar el atajo *Control+F5* que limpia la caché y actualiza la página. En Mac la solución es similar, pero usando el comando *cmd+shift+R*.

Por otro lado, este problema apareció también en teléfonos Android, en los cuales el mismo Telegram guardaba la cache de la ventana de navegador donde se ejecuta el juego, la solución fue una vez más limpiar la caché y volver a lanzar el juego.

Capítulo 7: Conclusiones del Proyecto

En este capítulo se van a presentar las conclusiones a las que hemos llegado tras la finalización del proyecto.

Tras varios años de carrera, empezamos a plantearnos cómo queríamos utilizar todo lo que habíamos aprendido en este tiempo. Para ello nos juntamos a debatir y nos pareció buena idea hacer un *bot* para Telegram, ya que es relativamente moderno y nos proponía un reto en nuestra trayectoria como estudiantes.

Nuestro objetivo era realizar un *bot* funcional que pudiese interaccionar con los usuarios; pero no todo estaba tan claro, hicimos varias pruebas con *Python*, bases de datos y con la API de Telegram, pero no nos convencía, ya que no veíamos claro como poder aplicar el potencial de todo lo aprendido a lo largo de la carrera y plasmarlo en un *bot*.

Estuvimos pensando en lo que realmente nos ilusionaba, hacer algo que llamase nuestra atención desde el primer momento, ahí llego nuestra idea, y la solución, un juego online, pero, ¿Por qué un juego online?

A la hora de desarrollar, te das cuenta que al final el escenario se limita siempre de alguna manera, por lo que creemos que un juego online es quizá la expresión más completa dentro de la programación web, ya que contiene la parte de aplicación como tal (Menú, modos de juego y el juego en sí), y a su vez, la parte de conexión (Cliente-Servidor) que ofrece una página web o un *web service*.

Podemos destacar el nivel alcanzado en programación *JavaScript* aplicado en el *web service* de Telegram *(Node.js)* utilizado como conector del bot, el *web service* para el servidor del juego *(Node.js)*, y también en el motor gráfico del cliente *(Phaser.io),* el cual se ha llevado gran carga de trabajo debido a la funcionalidad requerida. También cabe destacar la experiencia adquirida con los *WebSockets* ya que toda la funcionalidad multijugador está basada en los mismos.

Gracias a la universidad y a nuestro director de proyecto, hemos tenido la posibilidad de trabajar en remoto mediante un servidor en la universidad, el cual nos ha servido como repositorio Git y a la vez para hospedar los dos *web services,* dándonos la posibilidad de probar y modificar la aplicación multijugador en tiempo real desde el primer momento.

Pese a contar con la experiencia recogida tras estos años, ha sido un proceso duro de aprendizaje, de prueba y error constante, el cual nos ha hecho sacar lo máximo tanto individualmente como en equipo. Tras el esfuerzo de todo el grupo, los objetivos del proyecto quedan cumplidos, con un juego de aspecto atractivo y listo para jugar.

Project's Conclusions

In this chapter, we will show the conclusions that we have reached after the completion of this project.

After several years of career, we began to consider how we wanted to use everything we had learned so far. For this objective we met to discuss and we thought it was a good idea to make a Telegram bot, because it is relatively modern and it proposed us a challenge in our trajectory as students.

Our goal was to make a functional bot that could interact with users; but not everything was so clear, we did several tests with Python, databases and with the Telegram API, but we were not sure, since we were not clear about how to apply the potential of everything learned throughout the grade and condensate it into a bot.

We were thinking about what we were really excited about, do something that could caught our attention from the very first moment, then we got our idea, and the solution, an online game, but, Why an online game?

At the time of development, you realize that in the end the scenario is always limited in some way, so we believe that an online game is perhaps the most complete expression within the web programming, since it contains the application part as such (Menu, game modes and the game itself), and by the other side, the connections (Client-Server) offered in a web page or a web service.

We can highlight the level reached in JavaScript programming applied in the Telegram's web service (Node.js) used as the bot connector, the web service for the game server (Node.js), and also in the client's graphics engine (Phaser.io), which has taken a lot of work due to the functionality required, and also worth mentioning the experience gained with WebSockets since all multiplayer games functionality are based on them.

Thanks to the university and our project manager, we have been able to work remotely through a server in the university, which has served us as a Git repository and at the same time to host the two web services, giving us the possibility to test and modify the multiplayer application in real time from the beginning.

Despite having the experience gathered after these years, it has been a hard process of learning, with constant trial and error, that made us give the most individually and as a team. After the effort of the whole team, the objectives of the project were fulfilled, with an attractive looking game, and ready to be played.

Capítulo 8: Ampliaciones futuras

En este apartado se explican las posibles funcionalidades que se podrían añadir en un futuro al *bot*.

Más modos de Juego

Tal y como está estructurado el juego, es muy sencilla la implementación de nuevos modos. El servidor y el cliente tienen solo las dependencias necesarias, por lo que añadiendo un nuevo *.js* con la estructura del *track100m.js*, manteniendo las clases y variables y modificando el método *update()* con las reglas del nuevo modo, sería suficiente para añadirlo al juego.

Como acción futura, se podrían abstraer las reglas de cada juego para que solo hubiese que añadir un *.js* para los nuevos modos de juego y no tener que repetir el resto del código.

Mejora en las conexiones

El trato de conexiones actual es eficiente, pero podría mejorarse, ya que se ha dado el caso, en alguna de nuestras pruebas, de conexiones cruzadas. Este problema se solucionaría añadiendo en la caché del cliente su conexión actual, para que la retome cuando se desconecta inesperadamente o para que no se duplique si ejecuta el *bot* desde distintos dispositivos a la vez, con su misma cuenta.

Desvinculación de Telegram

Telegram actualmente sirve de registro del usuario y lanzadera del juego, pero se podría desvincular de Telegram y crear aplicaciones nativas para cada sistema operativo. Al tratarse de *JavaScript*, la tarea es sencilla, ya que bastaría con crear un contenedor web dentro de la aplicación.

El principal problema de desarrollar un juego en Telegram es seguir sus pautas de diseño, si no se siguen, no dejan que se publique en la plataforma de mensajería. Hay varias restricciones, pero las peores son la prohibición de guardar datos del usuario en una base de datos y el no poder utilizar sistemas de recompensas o logros. Hasta hace unas semanas tampoco se podían monetizar los *bots*, pero con la última actualización de mayo, se añadió el soporte de anuncios y *bots* de pago.

Future Extensions

This section explains the possible functionalities that could be added to the bot in the near future.

More game modes

As the game is structured, it is very easy to implement new game modes. The server and the client only have the necessary dependencies, so by adding a new .js with the structure of track100m.js, keeping the classes and variables and simply modifying the update() method with the new set of rules for the new mode, would be enough.

As a future action, we could abstract the rules of each game, so only adding a .js for the new game modes and not having to repeat the rest of the code.

Improved connections

The current connection handler is efficient, but there is still room for improvement, since in some of our tests, cross-connections have occurred. This problem could be solved by adding in the client's cache your current connection, so it will resume when unexpected disconnections occur or when you run the bot from different devices at the same time with the same account.

Unlinking the game from Telegram

Telegram currently serves us as a user registration and game launcher, but we could unlink the game from Telegram and create a native application for each operating system. Since we programmed using JavaScript, the task is simple, because it would be enough by generating a web container and launch the application in it.

The main problem developing a game in Telegram was following its design guidelines, if not followed, Telegram would not let the bot to be published on the messaging platform. There are several restrictions, but the worst is the prohibition of saving user data in a database and not being able to use a rewards or achievement system. Until a few weeks ago the bots could not be monetized, but with the last update of May, the ad support and payment methods were added.

Capítulo 9: Contribuciones personales

En este apartado cada participante va a hablar sobre sus aportaciones al proyecto, detallando su participación y las ayudas ofrecidas al resto de compañeros.

Se detallará todo en las siguientes partes: Conocimientos, Investigación, Pruebas, Desarrollo de la Aplicación y Memoria

Jesús de Oro García

Conocimientos

Como alumno del Grado en Ingeniería del Software, tengo conocimientos ampliados en el área de programación y algoritmos, sabiendo aplicar de una forma más eficiente patrones de diseño que hagan más sencillo el entendimiento del código desarrollado.

Cabe destacar mi experiencia previa en el tema de programación de videojuegos ya que he participado activamente en el desarrollo de un juego *online* masivo, aunque las tecnologías usadas en este proyecto sean totalmente diferentes.

Investigación

Primero empecé a buscar información sobre Telegram, más específicamente sobre el desarrollo de *bots*, y encontré que se podía desarrollar en multitud de lenguajes, pero nuestro tutor nos recomendó usar *Python* ya que había una API llamada *Telepot*, que es bastante sencilla, pero tiene una documentación muy escasa, ya que se trata de una API desarrollada por un chico en tu tiempo libre.

Una vez vimos más o menos como iba a ser nuestro *bot*, y que teníamos claro que iba a ser un juego, empecé a investigar HTML5, *JavaScript* y motores gráficos. En cuanto a motores gráficos, decidimos usar *Phaser.io,* estaba en auge y la gente hablaba muy bien de él.

Pruebas Previas

Mis primeras pruebas con los *bots* de Telegram fue desarrollar junto a mis compañeros un *bot* muy sencillo en Python, el cuál te pedía un tipo de restaurante que te apeteciese, y pasándole tu ubicación actual te mandaba un enlace a Google Maps con los restaurantes de esa comida más cercanos.

Después hice un *bot* individual más complejo; se trataba de una alarma, también en Python. A este *bot* le indicabas un evento a una hora y lo guardaba en la Base de Datos; cuando llegase la hora del evento, te avisa mandándote un mensaje privado.

Fue entonces cuando nos dimos cuenta de que no queríamos hacer un *bot* en el cual para interactuar con el tuvieses que mandar líneas de comandos por el chat, nos parecía una forma de interacción lenta y no apta para todos los públicos. Así que optamos por hacer un juego que conectase con Telegram, sabíamos que sería más tedioso, ya que

tendríamos que estar pasando información entre dos *web service*, pero nos motivaba ese reto.

Desarrollo de la Aplicación

El desarrollo de un juego en Telegram está dividido en tres partes, *bot* en Telegram (conector), Cliente del Juego y Servidor del Juego.

Decidimos desarrollar Héctor y yo conjuntamente el cliente-servidor, focalizándome yo en la parte de conexiones y de trata de información, y mi compañero en la lógica del juego y gráficos.

Utilizamos como motor gráfico *Phaser.io* y como gestor de conexiones *Socket.io*, buscando en la documentación de estas dos APIs, encontramos un ejemplo muy sencillo en el que había un monigote que iba donde tu pinchabas. Usando ese ejemplo como base, e implementando sobre él las conexiones con *Socket.io*, logramos tener a varias personas conectadas a el mismo mapa y poder ver el movimiento de cada uno de ellos, un juego muy tosco, pero era un comienzo.

Una vez completado el *bot* en Telegram y conectado con nuestro servidor, era capaz de leer los datos del usuario, ID del *chat* y Nombre. Con esa información se creaban *sockets* y el usuario disponía de un perfil en la aplicación.

El juego fue ganando en complejidad y era el momento de introducir los modos de juego. Empecé por el modo multijugador, donde después de varias reuniones acabamos pensando que la mejor forma de implementarlo sería con la creación de un *lobby* o sala de espera y varias *rooms* o habitaciones que se irían llenando según fuesen entrando los usuarios al modo multijugador.

La siguiente iteración del modo multijugador sería que los usuarios pudiesen jugar sus amigos, hasta entonces solo se podía jugar con todo el mundo. Utilizando la información obtenida del *bot* de Telegram, disponía del ID del *chat* del grupo en el que se había compartido el juego; usé ese ID como identificador de la habitación, de ese modo, solo los usuarios con un ID concreto podrían entrar en la sala de espera del grupo. Hubo que añadir una forma de iniciar la partida con menos de cinco personas, ya que puede haber grupos menos poblados.

De aquí en adelante quedaba pulir la aplicación e ir resolviendo problemas que nos íbamos encontrando poco a poco.

Destacar que no hay apenas juegos totalmente online en Telegram, por lo que no había absolutamente nada de información ni documentación y esto nos complicó bastante el desarrollo, pero no nos desmotivó.

Memoria

En cuanto a la memoria me he ocupado del apartado más técnico, ya que, al desarrollar el servidor, y con mis conocimientos de Ingeniería del Software, sabia explicarlo de una forma más comprensible que mis compañeros. También he colaborado en el apartado de Problemas y Soluciones que más me competían.

Héctor Gálvez Bernal

Conocimientos

Siendo alumno del Grado en Ingeniería Informática tengo conocimientos de programación y de patrones de diseño, y por optativas, he tratado de dirigir mi enseñanza hacia la programación web y las redes de datos.

A destacar mis conocimientos adquiridos en varios cursos de diseño web (HTML5, *JavaScript, Node.js, Angular.js*) y de editores gráficos como Photoshop.

Investigación

Llevo varios años usando Telegram a diario y utilizando *bots*, pero nunca me había picado la curiosidad de buscar información sobre cómo se hacían, hasta que nos confirmaron el trabajo de fin de grado.

Empecé a buscar documentación sobre *bots* de Telegram y de los distintos lenguajes de programación en los que se podían implementar, centrándome más en *Python,* ya que es el que recomendaba nuestro tutor, especialmente la API Telepot, desarrollada por *@nickoala.*

Después de varias reuniones con mis compañeros de grupo, decidimos que haríamos un juego en Telegram, ya que daba la casualidad de que se habían estrenado un par de meses antes de elegir el trabajo de fin de grado y sería una oportunidad de aprender algo muy nuevo. Además, al investigar y darme cuenta de que se programaban en lenguajes en los que yo había dedicado más tiempo (*JavaScript*, HTML5), me resultaría más sencillo ayudar a mis compañeros en los contratiempos que nos pudieran surgir durante la realización.

Por último, comencé a investigar sobre motores gráficos, y Jesús me recomendó *Phaser.io* para empezar a programar la lógica del juego.

Pruebas Previas

Nuestro primer acercamiento a los *bots* de Telegram fue programado en *Python*, y consistía en un *recomendador* de restaurantes en base a tu ubicación actual, pasando por el chat los comandos necesarios para interactuar con el *bot*.

Con ese *bot* aprendí la forma que tenía Telegram de enviar los metadatos de los mensajes, y empecé a probar el envío y recepción de mensajes filtrando por esos datos. Probé también los *bots* del resto de compañeros, haciendo mención a uno muy interesante, que creaba conversaciones anónimas entre dos personas a través del *bot*.

Habiendo visto cómo funcionaban los *bots* por línea de comandos, me di cuenta de que eso no es lo que quería hacer con nuestro *bot*, quería hacer algo más dinámico, más intuitivo. Y ahí es cuando me introduje en el mundo de los juegos de Telegram.

Busqué ejemplos de juegos en Telegram sin comandos, y encontré varios ya accesibles a través de la aplicación de mensajería, sencillos pero adictivos, y eso es lo que

queríamos buscar nosotros, aun sabiendo que no iba a ser fácil desarrollar el juego en Telegram ya que era muy reciente y no se encontraba nada de documentación por la red.

Desarrollo de la Aplicación

Después de hablar con mis compañeros para dividirnos el trabajo de la forma más eficiente, escogí centrarme en la lógica del juego e implementarla junto a Jesús a toda la parte del servidor.

Empecé a investigar sobre *Phaser.io* y encontré varias guías y ejemplos para empezar a probar y aprender su funcionamiento. Encontramos un juego sencillo donde el jugador se movía donde hacías *click*. No era nuestra idea de juego, pero nos sirvió para entender cómo funcionaba la arquitectura de *Phaser.io*, que es muy sencilla y tiene tres partes: *preload*, *create* y *update*.

Preload para cargar las imágenes necesarias en el juego, *create* para definir las variables del juego como los *sprites* de usuarios, tipo de físicas o tamaño del mapa, y el *update* que se ejecuta cada 20 milisegundos (por defecto) y tiene la lógica del juego y la interacción del usuario con él.

Una vez entendido el funcionamiento de *Phaser.io,* pude hacer una base del juego donde el usuario entraba, y pulsando dos botones, avanzaba de forma horizontal. Con la ayuda de Jesús, y añadiendo el servidor y el uso de *sockets,* pudimos añadir varios usuarios al mapa y poder verles moverse a la vez.

Jesús se siguió encargando de los modos de juego y de la parte online, mientras que yo me ocupaba del modo offline, que es el single, donde a parte de tu usuario se generan cuatro *bots* controlados por la máquina para competir contra ti.

Una vez que estaba la parte de la lógica del juego definida, faltaba darle personalidad. Busqué inspiración en varios juegos de móvil para ver qué elementos usar y cuáles eran las tendencias del momento. En base a eso diseñe los *sprites* de jugadores, el mapa, los botones, logos, todos los gráficos.

Memoria

Me he encargado de escribir la Definición del Sistema, explicando cómo funciona la aplicación y que se puede hacer con ella en un lenguaje sencillo y amigable.

También he completado el documento hablando de las futuras expansiones posibles y sobre como poder continuar con el proyecto.

Para concluir, le he dedicado tiempo al formato del texto, a la traducción y a dar coherencia al texto después de reunir los apartados de cada uno de los miembros.

Emilio Chico Muñoz

Conocimientos

Soy estudiante del Grado en Ingeniería Informática y tengo conocimientos en distintos lenguajes de programación y en organización de proyectos.

Investigación

Mirando los Trabajos de Fin de Grado, me encontré con este sobre *bots* de Telegram. En aquel momento no sabía ni que hacían los *bots*, ni siquiera había usado Telegram, pero me decidí a investigar y acabe maravillado por la cantidad de cosas que se pueden hacer: compartir *gifs*, mandar mapas, buscar videos en YouTube y mandarlos desde el chat, ver el tiempo, juegos y muchas cosas más.

Me introduje en el mundo de los *bots* de Telegram y estuve mirando los distintos lenguajes en los que se podía programar, nuestro tutor nos presentó *Python* como lenguaje de desarrollo. Ya lo había utilizado anteriormente en la carrera, y las primeras pruebas fueron creadas en este lenguaje.

Según fue avanzando el proyecto y cuando vimos que la API de *Python* no estaba optimizada para juegos, tuve que desecharla y pasarme a *JavaScript*, más concretamente a *Node.js* y una API para Telegram del usuario *@yagop*, muy similar a la de *Python* recomendada por el tutor, pero para *Node.js.*

Pruebas Previas

Para ir familiarizándonos con el entorno creamos un *bot* simple, pero que utilizaba todos los datos provenientes de los mensajes que el usuario le enviaba, y en función a esos datos, sugerirle restaurantes cercanos de la comida que ha especificado el usuario.

Este primer *bot* me sirvió para comprender el funcionamiento de paquetes en Telegram, aunque tuve que complementarlo también varios tutoriales en internet sobre *chats*, donde mejor se plasmaba dicho intercambio.

Después de esta prueba y vistos los diferentes *bots* de los demás compañeros, llegamos a la conclusión de que queríamos hacer algo más entretenido y dinámico que una simple interacción entre usuario y *bot* mediante comandos, por lo que nos decantamos por el desarrollo de un juego multijugador en tiempo real, sabiendo que iba a ser complicado debido a la poca información disponible y la escasez de ejemplos que nos permitieran enfocar la dinámica del juego.

El siguiente punto consistió en buscar algún juego que nos sirviera de ejemplo, no a nivel de funcionalidad, pero sí que nos ofreciera un patrón o algunas líneas para poder continuar y desarrollarlo a nuestro modo. Encontramos algunos que, aunque no nos sirvieran de gran ayuda incrementó nuestras ganas de superar este reto y seguir adelante con el trabajo.

Desarrollo de la Aplicación

Hicimos una reunión para dividirnos el trabajo, ya que vivo bastante más lejos que mis compañeros y de esta forma podíamos ir trabajando por nuestra cuenta.

Elegí hacerme cargo de la parte más cercana a Telegram, del conector. Se trata de un *bot* que recopila la información del usuario y la envía a través de un *callback* al servidor del juego.

Empecé siguiendo las pautas del profesor, y utilizando la API *Telepot*, una API en lenguaje *Python* con la que ya estuvimos haciendo pruebas, por lo que sería sencillo realizar este conector. Después de haber desarrollado la captura de información por parte del *bot*, tenía que generar el *callback* y sacarle al usuario el juego para jugar; problema, los juegos en Telegram eran algo muy nuevo y el desarrollador de la API en *Python* no había dado soporte a los juegos todavía. Por lo que me tocó empezar de cero buscando otra tecnología.

Hablé con mis compañeros y me dijeron que estaban usando *Node.js* para la lógica del juego, así que empecé a investigar por ahí y acabe encontrando la API de Telegram en *Node.js* que casualmente acababa de dar soporte a los *callback* de juegos.

Volví a reprogramar el *bot* en *Node.js*, y fui cambiado la salida del *callback* en función a lo que mis compañeros necesitaban. Primero pasándoles el ID de Telegram y el nombre del usuario, y más adelante el *Chat ID* para los modos multijugador.

Memoria

En el apartado de la memoria me he encargado de la parte de la Introducción, Planteamiento de trabajo y del Estudio de las tecnologías que hemos tenido en cuenta a la hora de reunir toda la información para comenzar nuestro proyecto.

También he participado en la selección de *Keywords* y en el Índice de ilustraciones con el que completamos gran parte de la investigación.

Anexo I: Referencias

[1] Crecimiento exponencial del uso del teléfono móvil:

Isabel F. Lantigua, (abril 2016). “El móvil supera por primera vez al ordenador para acceder a Internet”.

http://www.elmundo.es/sociedad/2016/04/04/57026219e2704e90048b465e.html

[2] Auge de las aplicaciones de mensajería:

Carlos Polo (enero 2017). “El auge de las aplicaciones de mensajería anticipa la era de los *chatbots*”.

https://es.linkedin.com/pulse/el-auge-de-las-aplicaciones-mensajer%C3%ADa-anticipa-la-era-carlos-polo

[3]Vivimos en un mundo saturado de información:

Antonio Ruíz-Giménez (mayo 2016). “La importancia de estar conectados en un mundo global”.

http://www.univision.com/noticias/opinion/la-importancia-de-estar-conectados-en-un-mundo-global

Anexo II: Bibliografía

Telegram:

https://telegram.org/

https://es.wikipedia.org/wiki/Telegram_Messenger

API Telegram:

https://core.telegram.org/api

Telepot API, autor: Nickoala:

https://github.com/nickoala/telepot

Node.js Telegram API, autor: Yagop:

https://github.com/yagop/node-telegram-bot-api

WhatsApp:

https://es.wikipedia.org/wiki/WhatsApp

http://www.informatica-hoy.com.ar/aprender-informatica/Que-es-Whatsapp.php

Line:

https://line.me/es/

https://es.wikipedia.org/wiki/LINE

Facebook Messenger:

https://www.messenger.com/

https://es.wikipedia.org/wiki/Facebook_Messenger

https://tecnologia.uncomo.com/articulo/como-funciona-la-app-facebook-messenger-26182.html

HTML5:

https://es.wikipedia.org/wiki/HTML5

https://developer.mozilla.org/es/docs/HTML/HTML5

http://es.html.net/tutorials/html/

JavaScript:

https://www.javascript.com/

https://es.wikipedia.org/wiki/JavaScript

https://developer.mozilla.org/es/docs/Web/JavaScript

Node.js:

https://nodejs.org/en/

https://www.ibm.com/developerworks/ssa/opensource/library/os-nodejs/

https://www.genbetadev.com/frameworks/como-funciona-node-js

http://www.netconsulting.es/blog/nodejs/

Node.js API:

https://nodejs.org/dist/latest-v6.x/docs/api/

Phaser.io:

https://es.wikipedia.org/wiki/Phaser

http://nicholls.co/blog/post/Creando-Juegos-HTML5-con-Phaser-en-Monaco

https://phaser.io/

Unity:

https://unity3d.com/es/unity

https://es.wikipedia.org/wiki/Unity_(motor_de_juego)

Websockets:

https://developer.mozilla.org/es/docs/WebSockets-840092-dup

https://es.wikipedia.org/wiki/WebSocket

https://www.html5rocks.com/es/tutorials/websockets/basics/

https://www.websocket.org/aboutwebsocket.html

Socket.io:

https://socket.io/

https://en.wikipedia.org/wiki/Socket.IO

Socket.io API cliente:

https://socket.io/docs/client-api/

Socket.io API servidor:

https://socket.io/docs/server-api/

Google Chrome:

https://es.wikipedia.org/wiki/Google_Chrome

https://www.google.com.mx/chrome/browser/desktop/?brand=CHBD&gclid=CjwKEAjwvYPKBRCYr5GLgNCJ_jsSJABqwfw7DuiLE4W0aHE7MswI3L9B0kqxCdusL2sPkrMEiFSaOhoC173w_wcB&dclid=CIaMpbTHvdQCFS4e0wodBVUMPw

https://developers.google.com/web/tools/chrome-devtools/console/

Git:

https://es.wikipedia.org/wiki/Git

https://git-scm.com/book/es/v1/Empezando-Fundamentos-de-Git

Sublime Text:

https://es.wikipedia.org/wiki/Sublime_Text